Universidad Carlos III de Madrid

Escuela Politécnica Superior
Departamento de Informática

Proyecto Fin de Carrera en Ingeniería en Informática

Sistema de control remoto por voz de los elementos de la carrocería de un vehículo BMW

Autor:
Francisco Olcina Grande

Tutor:
Javier Fernández Muñoz

9 de julio de 2012

A mi familia y a mis amig@s

que siempre han estado
ahí para mi.

«- Doc, oye, ¿me estás diciendo que has construido una máquina del tiempo... con un DeLorean?
- Yo creo que, si vas a construir una máquina del tiempo en un coche, ¿por qué no hacerlo con clase?»

Regreso Al Futuro - Robert Zemeckis.

Agradecimientos

Este es el fin de una etapa que ha durado muchos años, desde que me propuse hacer la Ingeniería en Informática hasta que la he terminado. Por el camino también terminé la Ingeniería Técnica en Informática de Gestión, pero no es lo mismo, no es lo que me propuse desde el comienzo, y como decía Yoda: «Hazlo o no lo hagas, pero no lo intentes».

Quiero agradecer en primer lugar a mis padres y a mi hermana el apoyo que siempre me han dado y que deseo que siga siendo así siempre. De la familia también hay alguien muy especial que me ha acompañado desde el comienzo dándome todo su cariño, y lo ha hecho... ¡¡sentándose sobre mi regazo!! Es mi gata Pulgui, que aprovecha cada vez que me siento en mi silla para subirse encima. Creo que la mitad del título le pertenece...

También quiero agradecérselo a las siguientes personas:

A mis compañeros y amigos de la Universidad Carlos III, en especial a la gente de ARCOS. Con ellos pasé muchos años mientras estudiaba y trabajaba en la misma universidad, y me encanta mirar hacia atrás y rememorar todas las "quedadas" que hemos hecho.

A mis compañeros y amigos de la Universidad Rey Juan Carlos, tanto los que he tenido cuando estaba en Libresoft, como a los actuales.

A mis amigos de siempre y a los que he ido haciendo estos últimos años, a ellos les debo una gran parte de mi felicidad.

Y por último, a las chicas "especiales" que al conocerlas me han ido cambiando la vida (para bien claro jeje), y tienen un lugar muy importante en mi corazón.

Tod@s ell@s saben quienes son, y yo se quienes son ell@s, esta vez no me mojo jaja.

Para todos vosotros, y enormemente agradecido,

Paco.

Resumen

A finales de los años 70, los vehículos comenzaron a incorporar sistemas electrónicos que comprobaban el correcto funcionamiento del motor. Éstos sistemas electrónicos evolucionaron llegando a controlar practicamente la totalidad de los elementos que componen un vehículo, no sólo obteniendo información sobre su funcionamiento mediante sensores, sino modificándolo mediante actuadores.

Ésta electrónica está repartida por el interior de los vehículos mediante centralitas, cada una dedicada a un propósito específico. Las centralitas permiten conexiones con sistemas externos al vehículo, denominados sistemas de diagnosis, y a través de los cuales se puede recopilar información de las propias centralitas o solicitar una acción específica sobre los elementos que controlan.

El presente proyecto desarrolla un sistema para controlar de forma remota las operaciones que puede realizar una centralita que gestiona los elevalunas, el techo eléctrico y el cierre centralizado (entre otros elementos) haciendo uso de las funciones de diagnosis.

El sistema se compone de un circuito impreso conectado al vehículo que sirve de interfaz de conexión remoto y de una aplicación para móviles con S.O Android que se comunica con el interfaz vía Bluetooth.

Palabras clave: vehículo, centralita, diagnosis, circuito, Android, Bluetooth.

Abstract

In the late 70's, vehicle manufacturers started assembling electronic systems to check the correct engine operation. These electronic systems evolved to gain control of almost all the components of a vehicle, These electonic systems were not only used to obtain the information collected by the sensor but also, to modify de behaviour with the actuators.

This electronic is distributed along the inner parts of the vehicles and the control is also distributed along several control units, each one dedicated to a specific purpose. These control units can comunicate with outside systems, called diagnostic systems. A diagnostic system can collect information from the control units or request a specific action to be perfomed by some component that is controlled by this contol unit.

This project develops a system that allows the user to control remotely several components of the vehicle operations including the windows, the sunroof and the central locking (among other things) using the diagnostic functionality of the vehicle.

The system is composed by a printed circuit connected to the vehicle which acts as a remote connection interface and a mobile application for Android OS that communicates with the interface via Bluetooth.

Keys: vehicle, control unit, diagnostics, circuit, Android, Bluetooth

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción y objetivos

1.1. Introducción

El presente proyecto presenta un sistema que permite controlar de forma remota diferentes elementos de un vehículo mediante un móvil con sistema operativo Android.

Los elementos del vehículo controlados remotamente son: los cierres de las puertas, los elevalunas eléctricos, el techo eléctrico y el limpiaparabrisas. El control se efectúa por voz mediante una aplicación desarrollada en Android ejecutándose en un móvil con conexión a internet.

El sistema tiene un enfoque práctico: se persigue que el conductor del vehículo active los elementos mencionados anteriormente por voz, mejorando el confort en la conducción, y aumentando la seguridad al evitar distracciones buscando los mandos de accionamiento.

1.2. Antecedentes

En el vehículo utilizado para el presente proyecto se probó con éxito un sistema de diagnosis compuesto de un software instalado en un portátil y un hardware que se conectaba al vehículo. El sistema permitía, entre otras funciones, gestionar desde el portátil los cierres de las puertas, los elevalunas eléctricos, el techo eléctrico y el limpiaparabrisas.

El sistema de diagnosis utilizado fue Carsoft de la compañía Carsoft International, presentado más en detalle en el capítulo del Estado de la Cuestión (ver 2.3.3 en página 17). Al contemplar con éxito como se gestionaban los elementos del vehículo desde el portátil, se pensó en la viabilidad de crear un sistema nuevo compuesto de un elemento receptor Bluetooth conectado permanentemente al vehículo, y una aplicación para móviles que se comunicase con el elemento receptor Bluetooth. Desde la aplicación se accionarían los elementos que podía gestionar

Carsoft (elevalunas, cierres, etc.), pero en esta ocasión, accionados por comandos de voz.

1.3. Objetivos

El objetivo principal del presente proyecto es construir un sistema que permita el control remoto de varios elementos del vehículo. El sistema se compone de un interfaz remoto conectado al vehículo y una aplicación para móviles con S.O Android.

Para la elaboración del sistema se persiguen los siguientes objetivos secundarios:

- Conocer las características de la electrónica del vehículo del proyecto.
- Conocer las características de la comunicación con los elementos electrónicos del vehículo necesarios para el desarrollo del sistema.
- Construir un circuito que permita comunicarse con el vehículo.
- Conocer la arquitectura del S.O Android, y adquirir las habilidades necesarias para desarrollar la aplicación que forma parte del sistema.

1.4. Fases del proyecto

En la primera fase del proyecto se realizó un análisis para especificar lo que debe realizar el sistema. En dicho análisis se obtuvieron una serie de requisitos y de casos de uso que sirvieron para establecer el trabajo a realizar a continuación.

En la siguiente fase se realizó un estudio de la electrónica del vehículo con objeto de hallar los elementos que gestionan los elevalunas, cierres, etc. De dicho estudio se obtuvo que sólo una centralita era la encargada de gestionar los elementos mencionados.

Tras el estudio de la electrónica, se creó un entorno de pruebas sobre una mesa de trabajo con la centralita hallada en la fase anterior. Utilizando éste entorno y distintas soluciones de diagnosis se pudo realizar un estudio más exhaustivo de la comunicación con la centralita. Aplicando ingeniería inversa sobre los diálogos establecidos entre las soluciones de diagnosis y la centralita, se consiguieron los códigos que controlan la acción de elevalunas, cierres, etc.

Tras el análisis exhaustivo de la comunicación con la centralita, se desarrolló el un circuito cuyo objetivo es servir de interfaz de conexión remota vía Bluetooth al vehículo. Éste circuito contiene entre otros componentes un microcontrolador que tuvo que ser programado con un código creado para la ocasión.

Habiendo finalizado el desarrollo del circuito, se procedió a desarrollar la apli-

cación para móviles con S.O Android. En esta fase se realizó un estudio de la programación sobre dicho S.O, incluyendo el estudio de los medios necesarios para integrar en la aplicación las tecnologías de reconocimiento de voz, síntesis de voz, y Bluetooth. Una vez comprobada la viabilidad de integrar las tecnologías anteriores en la aplicación se procedió a su desarrollo.

La última fase del proyecto consistió en la integración del interfaz de conexión remota sobre el vehículo y la realización de diversas pruebas. Se comprobó que la integración de la aplicación desarrollada en el S.O Android con el circuito conectado al vehículo cumplía los objetivos marcados.

1.5. Medios empleados

Los medios empleados para el desarrollo del proyecto fueron los siguientes:

- **Vehículo**: se contó con un vehículo BMW Serie 5 E34 del año 1993.
- **Soluciones de diagnosis**: se utilizaron las soluciones de diagnosis Carsoft 6.5 y Ediabas ToolSet32. Éstas soluciones se componen de un software y un interfaz hardware que se conecta al vehículo y al equipo donde esté instalado el software. En el caso de Ediabas ToolSet32, sólo se contaba con el software, por lo que hubo que construir un interfaz hardware compatible.
- **Centralita**: se realizaron pruebas con una centralita obtenida de desguace equivalente a la que tiene el vehículo integrada.
- **Mesa de trabajo**: se contó con la mesa de trabajo para los desarrollos electrónicos. En la mesa se tuvieron elementos como: placas de desarrollo (*protoboards*), semiconductores, herramientas para cables, y una fuente de alimentación de +12V.
- **Portátil**: se contó con un portátil modelo Acer Aspire One Happy para realizar pruebas sobre el vehículo.
- **Ordenador de sobremesa**: se contó con un ordenador de sobremesa Pentium IV del que se aprovechó su puerto COM1 local, puerto del que carecía el portátil.
- **Teléfono móvil**: se contó con un móvil Sony Ericsson Xperia Neo V con S.O Android integrado, para el desarrollo e implantación de la aplicación en Android.
- **Componentes electrónicos**: se contó con diversos componentes electrónicos y material para realizar circuitos impresos. Estos materiales están listados en el capítulo 4 (ver 5.3.1 en página 103).

1.6. Estructura de la memoria

La memoria se estructura en los siguientes capítulos:

- **Capítulo 1 - Introducción y objetivos**: donde se exponen los antecedentes, objetivos, fases, medios empleados y la estructura del presente proyecto.
- **Capítulo 2 - Estado de la cuestión**: donde se expone toda la información sobre las tecnologías involucradas en la solución que se quiere alcanzar. Se abordan los siguientes temas: electrónica en vehículos, electrónica y comunicaciones en los vehículos BMW y específicamente en el vehículo el proyecto, la transmisión en serie, los microcontroladores, la tecnología Bluetooth y el S.O Android.
- **Capítulo 3 - Análisis del sistema**: donde se realiza un análisis de lo que debe realizar el sistema. El análisis se compone de una serie de requisitos y de casos de uso, que sirven para establecer el trabajo a realizar en el desarrollo del proyecto.
- **Capítulo 4 - Estudio de la comunicación con el vehículo**: donde se realiza un estudio de la comunicación con el vehículo utilizando la metodología de ingeniería inversa.
- **Capítulo 5 - Interfaz de conexión remota**: donde se muestra la construcción del circuito que sirve de interfaz remoto.
- **Capítulo 6 - Aplicación en Android**: donde se exponen todas las fases de desarrollo de la aplicación para el S.O Android.
- **Capítulo 7 - Integración y pruebas**: donde se expone la integración del interfaz de conexión remota con el vehículo y las pruebas realizadas sobre todo el sistema.
- **Capítulo 8 - Conclusiones y trabajos futuros**: donde se exponen las conclusiones extraídas después del trabajo realizado y los trabajos futuros, tanto de mejora como de ampliación de funcionalidades.
- **Capítulo 9 - Planificación y presupuesto**: donde se presenta una planificación y un presupuesto para la elaboración de todo el proyecto.
- **Anexo A**: donde se incluye el código del microcontrolador AVR utilizado en el proyecto.
- **Glosario**: donde se incluye un listado de términos y definiciones específicos del proyecto.
- **Bibliografía**: donde se exponen las referencias utilizadas durante el desarrollo del proyecto.

Capítulo 2

Estado de la cuestión

2.1. Introducción

En éste capítulo se da una visión general sobre las tecnologías y elementos necesarios para el desarrollo del proyecto. Las secciones en las que se divide el capítulo son las siguientes:

- **Diagnosis en vehículos**: esta sección cubre los aspectos sobre la electrónica de los vehículos y la comunicación y diagnosis que se puede realizar sobre ellos.
- **BMW**: esta sección da una visión general de la electrónica utilizada en los vehículos de la marca BMW y una visión más específica sobre el vehículo utilizado para el presente proyecto.
- **Transmisión en serie**: esta sección detalla el funcionamiento de las comunicaciones en serie.
- **AVR**: esta sección trata sobre los microcontroladores y específicamente los AVR de la compañía Atmel.
- **Bluetooth**: esta sección trata sobre el estándar de comunicaciones Bluetooth.
- **Android**: esta sección da una visión general sobre la plataforma Android y el sobre el desarrollo de aplicaciones para la plataforma.

2.2. Diagnosis en vehículos

La diagnosis en los vehículos es un proceso en el cual se establece una comunicación con el equipo electrónico de un vehículo para obtener información del mismo. Además de éste objetivo principal, algunos sistemas de diagnosis tienen un papel más activo permitiendo el control de los elementos del vehículo, así como

la reprogramación del software o la memoria de las centralitas.

A continuación se hará una breve exploración sobre la historia de la electrónica en los vehículos y después se explicarán dos de los estándares de diagnosis más importantes: el OBD I y el OBD II.

2.2.1. Electrónica en vehículos

A finales del siglo XIX se introdujo en Europa el automóvil como medio de transporte, estos primeros vehículos llevaban un motor de combustión interna de cuatro tiempos bastante pesado y rudimentario. Mas adelante, Daimmler ideó una variante mucho mas ligera que sería el precursor de todos los motores a explosión posteriores. Con los años, los automóviles fueron incorporando innovaciones que aumentaron su rendimiento y mejoraron sus prestaciones, estas mejoras incluían el uso de diferencial, correas, baterías, etc. Pero en su diseño, el motor de combustión interna no experimento cambios substanciales.

Ya bien entrado el siglo XX las innovaciones mecánicas siguieron sin afectar al diseño básico de de los motores, suponiendo tan solo la adicción de elementos orientados a la optimización de los mismos. Es a finales de los 70 cuando se empieza a incorporar la electrónica a los automóviles. Se añadieron los primeros sensores a los motores para verificar su correcto funcionamiento, también se añadieron unidades de control del motor que manejaban dichos sensores. El objetivo inicial de estos elementos electrónicos era el control del las emisiones de gases contaminantes y facilitar la diagnosis de averías.

A partir de la década de los 80 la mayor parte de las innovaciones provienen principalmente de la incorporación de la electrónica, y no de la incorporación de mejoras mecánicas. Se añadieron multitud de sensores y se fueron mejorando las unidades de control del motor. Hoy en día un automóvil puede incorporar mas de 200 sensores y más de una unidad de control. Hay unidades de control para el motor, climatizador, airbag, etc.

Las primeras unidades de control eran Módulos de Control de Motor o ECM (*Engine Control Module*), con el tiempo estas ECMs se hicieron más complejas y pasaron a convertirse en Unidades de Control Electrónico o ECUs (*Electronic Control Unit*) más conocidas en español como centralitas.

Actualmente los sensores se encargan de la medición de temperaturas, presiones, rotaciones, volúmenes, y multitud de parámetros de funcionamiento. La información que captan los sensores es enviada y almacenada en las centralitas. Toda esta información permite que el propio automóvil "conozca" su estado. En realidad, los sensores se limitan a detectar una serie de valores que envían a la centralitas y una vez allí son comparados con los valores óptimos que están almacenados en las memorias. Cuando se encuentra un valor incorrecto, la centralita notifica un fallo avisando al conductor de alguna forma (indicadores luminosos, sonidos, etc), los fallos quedan almacenados para su posterior verificación por un

mecánico. En algunas ocasiones, las centralitas simplemente fallan sin notificar ningún error.

Cuando un coche sufre una avería, el taller mecánico usa un escáner para conectarse con las centralita del vehículo. Un escáner es un dispositivo que se conecta a una centralita para acceder a los datos que ésta tiene almacenados en la memoria. Éstos datos incluyen los fallos que se han producido además de otros valores [16].

2.2.2. OBD (*On Board Diagnostics*)

Orígenes

Para combatir los problemas de polución en Los Ángeles, el Estado de California exigió sistemas de control de emisiones de gases en los modelos de automóvil posteriores a 1966. El Gobierno Federal de los Estados Unidos extendió estos controles a toda la nación en 1968. El Congreso aprobó el *Clean Air Act* (Acta Antipolución) en 1970 y creó la Agencia de Proteccion Medioambiental o EPA (Environmental Protection Agency). La EPA inició el desarrollo de una serie de estándares en la emisión de gases y unos requerimientos para el mantenimiento de los vehículos con el fin de ampliar su vida útil.

Para cumplir estos estándares los fabricantes implementaron sistemas de encendido y de alimentación controlada de combustible, con sensores que median las prestaciones del motor y ajustaban los sistemas para conseguir una mínima polución. Estos sensores también permitían una cierta ayuda en la reparación.

Al comienzo eran pocas normas y cada fabricante tenía sus propios sistemas y señales. En 1988, la Sociedad de Ingenieros Automotrices (SAE) propuso un conector estándar y un conjunto de señales de test de diagnóstico [10].

OBD I

On-Board Diagnostics I (OBD I) fue la primera regulación de OBD que obligaba a los productores a instalar un sistema de monitorización de algunos de los componentes controladores de emisiones en automóviles. Obligatorios en todos los vehículos a partir de 1991, los sistemas de OBD I no eran tan efectivos porque solamente monitorizaban algunos de los componentes relacionados con las emisiones, y no eran calibrados para un nivel específico de emisiones [9].

OBD II

En 1994, un organismo estatal de California, el CARB (California Air Resources Board), aprobó la especificación OBD II convirtiéndola en requisito legal para

todos los automóviles y camiones ligeros nuevos del Estado de California. A partir de 1996 se convirtió en un requisito legal en todo EEUU.

En Europa se introdujo el OBD ajustándose al OBD II americano; según la Directiva Europea 98/69EG, los coches a gasolina del año 2000 en adelante, los coches a diésel del año 2003 en adelante y los camiones desde el año 2005 en adelante tienen que estar provistos de un OBD II.

El estándar OBD II especifica que el Modulo de Control de Motor o ECM (Engine Control Module) debe monitorizar ciertos componentes del vehículo relacionados con las emisiones de gases para asegurar un correcto funcionamiento, y que se ilumine una Luz Indicadora de Fallo o MIL (*Malfunction Indicator Lamp*) en el cuadro de instrumentos cuando se detecta un problema (ver figura 2.1). El sistema OBD también aporta un sistema de Códigos de Error de Diagnostico o DTC (*Diagnostic Trouble Codes*) y unas tablas de errores en los manuales de reparación para ayudar a los mecánicos a determinar las causas mas probables de avería en el motor y problemas en las emisiones.

Figura 2.1: Luz Indicadora de Fallo o MIL(Malfunction Indicator Lamp)

Para aplicar los estándares ODB, OBD II y otros que han aparecido a lo largo del tiempo (EOBD[1], JOBD[2], etc.), se utilizan protocolos de comunicación tales como las normas ISO9141, ISO9141-2, ISO14230 (KWP2000), SAEJ1850, SAEJ1979, CAN BUS y VAN BUS entre otros. Algunos protocolos han sido definidos por ISO o SAE, otros son implementaciones propietarias de algunos fabricantes (implementaciones propietarias son sistemas propios de cada fabricante que no constituyen estándares en la industria y que solamente pueden ser empleados por dichos fabricantes, por ejemplo Mercedes, BMW o VAG) pero todos estos protocolos cumplen con las especificaciones OBD, OBD II, etc.

En la figura 2.2 se observa el conector instalado entre los pedales y el volante de un vehículo, en la figura 2.3 aparece la función de cada terminal del conector.

[1]EOBD es la abreviatura de *European On Board Diagnostics* (Diagnóstico de a Bordo Europeo), la variación europea de OBD II.

[2]JOBD es la versión de OBD II para vehículos vendidos en Japón

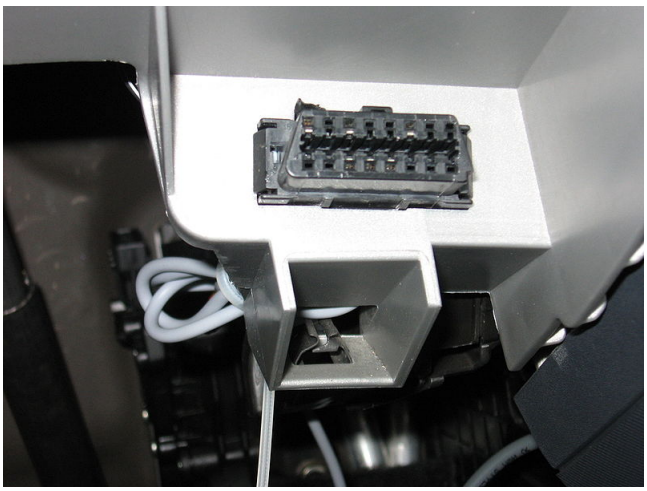
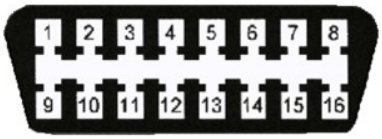


Figura 2.2: Conector OBD II



1 – Sin uso	9 – Sin uso
2 - J1850 Bus positivo	10 - J1850 Bus negativo
3 – Sin uso	11 – Sin uso
4 - Tierra del Vehículo	12 – Sin uso
5 – Tierra de la Señal	13 – Tierra de la señal
6 - CAN High	14 - CAN Low
7 - ISO 9141-2 - Línea K	15 - ISO 9141-2 - Línea L
8 – Sin uso	16 - Batería - positivo

Figura 2.3: Terminales del conector OBD II

2.3. BMW

BMW (siglas en alemán de: *Bayerische Motoren Werke*, "Fábricas bávaras de motores") es un fabricante alemán de vehículos de gama alta y de motocicletas.

Figura 2.4: Logotipo de la compañía BMW.

Los modelos de vehículos fabricados por BMW se catalogan en series, orientadas a un tipo de segmento de mercado. Así pues, BMW consta de las siguientes series principales:

- **Serie 1**: son vehículos del segmento C (conocidos como compactos).
- **Serie 3**: son vehículos del segmento D (conocidos como sedán)
- **Serie 5**: son vehículos del segmento E (conocidos como berlina grande).
- **Serie 7**: son vehículos del segmento F (conocidos como berlina de lujo).
- **Serie Z**: son vehículos denominados *roadsters*. Se subdividen a su vez en las series Z1, Z3, y Z4.
- **Serie X**: son vehículos de tipo 4x4. Se subdividen a su vez en función de su tamaño en X1, X3, X5 y X6.

BMW asigna a cada tipo de vehículo un código de carrocería según su serie y su generación. Por ejemplo, un vehículo BMW serie 5 de tercera generación (años 89-95) tiene un código E34, y un vehículo BMW serie 5 de sexta generación tiene un código F10.

2.3.1. Buses de datos en los vehículos BMW

Hasta la introducción en el mercado del modelo E31 de BMW, la información de cada centralita se enviaba y recibía por un cable independiente, provocando un aumento del cableado del vehículo según aumentaba el número de centralitas.

La necesidad de guiar cada cable por separado, y de encontrar espacio en la carrocería para alojarlos, supuso un problema para la producción y para la fiabilidad y detección de fallos. La solución a éste problema fue utilizar una arquitectura de bus, donde se encontraron los siguientes beneficios:

- Más fiabilidad al reducir el número de cables, conectores y componentes.

- Reducción de los conectores de las centralitas, al decrecer el número de interfaces entre las centralitas a uno o dos cables.
- Flexibilidad de la configuración del sistema y futuras aplicaciones.
- Reducción de costes en componentes, embalajes y detección de fallos.

En la actualidad los vehículos BMW tienen varios buses de datos, divididos en grupos de centralitas que manejan datos en común. Éstos buses son los siguientes:

- **D-bus**: es el bus más antiguo utilizado en los vehículos BMW y el que se va a utilizar en éste proyecto. En el siguiente apartado se hará una explicación más detallada sobre éste bus.
- **I-bus**: se trata de un bus de comunicaciones en serie que suele nombrarse junto al K-bus, denominándose ambos I/K bus o *Body Bus*. Son técnicamente iguales pero enfocados a distintos grupos de centralitas. El I-bus esta sobre todo dirigido al contenido multimedia (navegador, radio, etc.).
- **K-bus**: se trata de un bus de comunicaciones en serie que junto al I-bus, forma lo que se denomina I/K bus o *Body Bus*. En el caso del K-bus, está dirigido a centralitas como la de climatización automática (IHKA) o el inmovilizador del vehículo (EWS).
- **P-bus**: es un bus de comunicaciones en serie dirigido exclusivamente a los vehículos equipados con la centraliza ZKE III, encargada de las ventanas, el cierre, y el techo del vehículo entre otros usos.
- **M-bus**: es un bus de comunicaciones serie utilizado exclusivamente para que la centralita de climatización automática (IHKA) active los motores que abren y cierran las aperturas de aire del habitáculo.
- **CAN-bus**: es un bus de comunicaciones en serie utilizado para todas las centralitas que dependan del motor del vehículo. Entre otras está la centralita de inyección (DME), la de control de estabilidad (DSC) y la de electrónica del motor (EML).

D-bus (*Diagnostic Bus*)

El D-bus es un bus de comunicaciones en serie que permite realizar conexiones entre el equipo de diagnosis y las centralitas del vehículo. La centralita con la que se quiere establecer una conexión, es seleccionada mediante el envío de una trama de diagnosis que incluye la dirección de la centralita en cuestión. Por petición del equipo de diagnosis, la centralita puede transmitir información y el contenido de su memoria de errores o bien activar una salida digital. El D-bus está sólo activo cuando el equipo de diagnóstico está conectado a él.

El D-bus es el bus de comunicaciones más antiguo usado en los vehículos BMW, fue introducido en 1987 como TXD y permitía la comunicación entre la centralita

de inyección (DME), y un equipo de diagnosis conocido como SUN 2013 Service Tester.

Los modelos más antiguos utilizaban una segunda línea denominada RXD que permitía al equipo de diagnosis establecer la comunicación con las centralitas. Hasta el año 2001, los vehículos BMW tenían en el vano del motor un conector de diagnosis de 20 pines con las dos líneas siguientes operando a 9.600 baudios por segundo:

- RXD en el pin número 15.
- TXD en el pin número 20.

Algunas centralitas a partir del año 1997 no requerían la línea RXD para establecer una comunicación con el equipo de diagnosis, por ello es frecuente que muchos vehículos no tuviesen habilitado el pin número 15 en sus conectores de diagnosis de 20 pines.

Por otro lado, para satisfacer los requerimientos del OBD II, a partir de 1995 el conector estándar de OBD II fue instalado en el interior de todos los vehículos. Éste conector daba acceso a todas las centralitas utilizando cualquier herramientas de diagnostico que cumpliese los estándares del OBD II [8].

En último lugar, según la información recopilada a través de internet, el protocolo que se utiliza en la comunicación del D-bus se denomina DS-2, cuyas tramas se construyen de la siguiente forma [12]:

```
[Address] [Length] [Data1] [Data2] ... [Data n] [Checksum]
```

La cabecera de la trama incluye la dirección de la centralita a la que va dirigido, después existen unos bytes con la longitud seguidos de los bytes de datos, y en la última posición un byte o bytes de suma de comprobación (*checksum*).

Interfaces de diagnóstico (ADS y OBD)

Los equipos de diagnosis compatibles con los vehículos BMW incorporan un módulo hardware que sirve de interfaz y por tanto conecta el equipo de diagnosis con el conector de diagnosis del vehículo. Existen dos tipos de interfaz para los equipos de diagnosis:

1. **ADS**: (del Alemán *Aktive DiagnoseStecker*) éste interfaz es el utilizado en los vehículos más antiguos, equipados con el conector OBD I de 20 pines con líneas RXD y TXD.
2. **OBD**: éste interfaz permite conectarse a los vehículos más modernos, equipados con conectores OBD II, o bien con conectores OBD I que sólo disponen de la línea TXD.

2.3.2. BMW Serie 5 E34

Los vehículos BMW serie 5 con código de carrocería E34 se fabricaron desde el año 1989 hasta el año 1995. La figura 2.5 muestra el vehículo utilizado para el presente proyecto, en concreto una unidad de E34 con las siguientes características:

- **Denominación comercial**: 525i, versión US (estadounidense).
- **Motor**: código M50B25, 2.500 cc, 6 cilindros, 24 válvulas, inyección de gasolina.
- **Potencia**: 192 CV.
- **Color**: granate (denominación comercial *Calypso Rot*).
- **Código VIN**: GB35839.
- **Extras de serie**:
  - Elevalunas eléctricos.
  - Techo solar.
  - Airbag del conductor.
  - Dirección asistida.
  - Calefacción y aire acondicionado.
  - Ordenador de abordo.
  - Control de velocidad de crucero (*tempomat*).
  - Asientos eléctricos.
  - Cuadro con información digital.
  - Espejos calefactables.
  - Interior de cuero.
- **Extras añadidos posteriormente**:
  - Alarma y cierre centralizado con mando.
  - Faros *Angel Eyes*.

Conector de diagnosis del E34 (OBD I)

El vehículo del proyecto posee un conector de diagnosis de tipo OBD I. Éste conector esta alojado en el vano del motor y tiene forma de conector redondo de 20 pines. La flecha verde de la figura 2.6 indica la localización del conector en el vehículo.

Figura 2.5: Vehículo del proyecto (BMW Serie 5 E34).

En la figura 2.7 se observa el esquema del conector y la numeración de cada pin. El vehículo del proyecto tiene habilitados los pines: 1, 7, 12, 14, 15, 16, 18, 19 y 20. La tabla 2.1 muestra la función de cada uno de los pines anteriores.

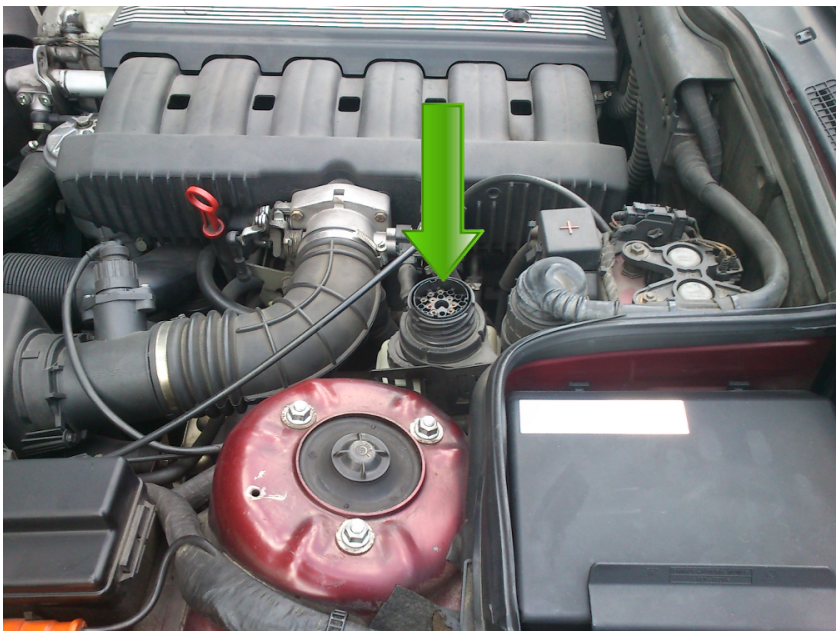


Figura 2.6: Localización del conector OBD I en el vano del motor.

Figura 2.7: Conector OBD-I de 20 pines de BMW.

Pin	Función
1	Señal del sensor de velocidad
7	Reset del intervalo de servicio
12	Indicador de carga del alternador
14	+12V en todo momento
15	Línea RX de datos
16	+12V con contacto en posición 2
18	Línea de programación
19	Tierra
20	Línea TX de datos

Cuadro 2.1: Función de los pines del conector OBD-I.

Centralita ZKE

La centralita ZKE (del alemán *Zentrale Karosserie Elektronik*), conocida también como *Central Body Electronics* o *General Module* es la encargada de gestionar diversas funciones del habitáculo del vehículo BMW. El vehículo del proyecto lleva integrado la primera versión de ésta centralita (ver figura 2.8), que gestiona los siguientes elementos:

- Elevalunas eléctricos.
- Cierre centralizado.
- Techo eléctrico.
- Limpiaparabrisas (velocidad y presión contra la luna delantera).

- Calefacción de la cerradura del conductor.
- Bomba del limpiaparabrisas.
- Luz interior.
- Luces de emergencia.

Figura 2.8: Vista de frente y desde arriba de la centralita ZKE.

La centralita está ubicada bajo el asiento trasero del lado del conductor, como indica la flecha verde de la figura 2.9.

Figura 2.9: Localización de la centralita ZKE en el vehículo.

Como el resto de centralitas del vehículo, está conectada al D-bus y tiene la capacidad de activar cualquiera de los elementos anteriores mediante ordenes enviadas a través del bus. Como se verá más adelante, las soluciones de diagnosis Carsoft y Ediabas ToolSet permitirán realizar vía D-bus la activación de los componentes que gestiona la centralita ZKE.

2.3.3. Carsoft

Carsoft es una solución de la compañía Carsoft International para la diagnosis de vehículos BMW, Mini y Mercedes. La solución para BMW, se compone de los siguientes elementos:

- Un CD con el software Carsoft para el sistema operativo Microsoft Windows.
- Un interfaz hardware que conecta el vehículo al PC donde esté instalado el software.
- Un cable serie DB9, para conectar el PC al interfaz hardware.
- Un conector OBD II para conectar el interfaz hardware a los vehículos BMW que dispongan de dicho conector.
- Un conector OBD I de 20 pines para conectar el interfaz hardware a los vehículos BMW que dispongan de dicho conector.

La figura 2.10 muestra el conjunto de Carsoft para BMW.

Figura 2.10: Solución de Carsoft para BMW.

El software de Carsoft para vehículos BMW permite realizar un diagnóstico de cualquiera de las centralitas del vehículo mientras estén soportadas, realizar algunas reprogramaciones e incluso activar alguno de los componentes que gestionan las centralitas. En la figura 2.11 se muestra como Carsoft es capaz de activar varios de los componentes que gestiona la centralita ZKE.

El interfaz hardware está diseñado para detectar las centralitas que contiene el vehículo, para ello utiliza dos elementos programables, un microcontrolador Atmel y un CPLD (Circuito Lógico Programable Complejo) de XILINX (ver figura 2.12). Éstos elementos dificultan la reproducción del interfaz al contener un firmware protegido contra lectura.

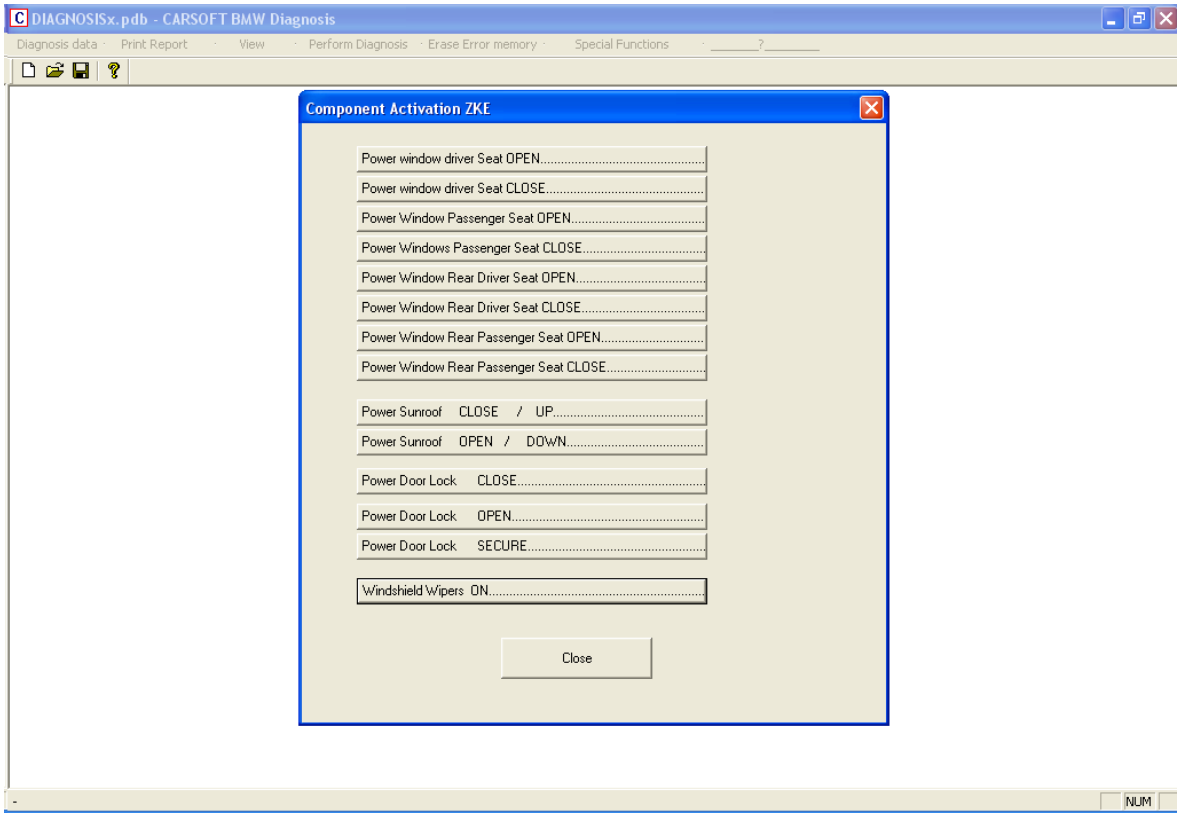


Figura 2.11: Ventana de operaciones sobre la centralita ZKE en Carsoft.

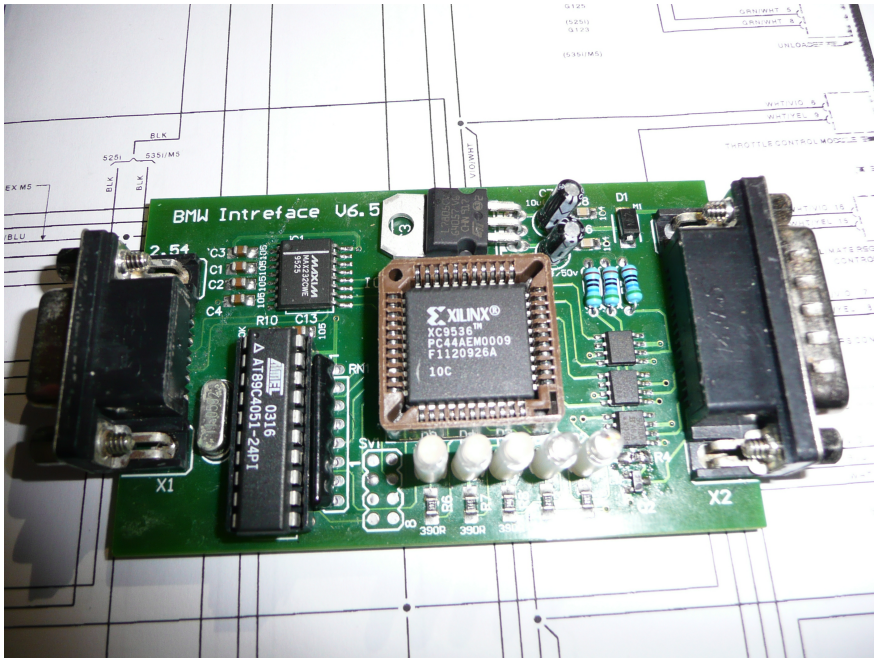


Figura 2.12: Interior del interfaz hardware de Carsoft para BMW.

2.3.4. EDIABAS ToolSet 32

La compañía BMW tiene su propio software de diagnóstico enfocado a sus vehículos. Éste software es un conjunto de aplicaciones que tienen un driver en común denominado EDIABAS (del inglés *Electronic Diagnostic Basic System*)

EDIABAS funciona como un intérprete entre las aplicaciones de BMW y las centralitas de los vehículos. Para ello utiliza ficheros denominados *ECU description files* o SGBDs (del alemán *SteuerGeräteBeschreibungsDatei*) que contienen las tramas de bytes necesarias para dialogar con cada tipo de centralita. Además, EDIABAS provee un API para distintas plataformas (UNIX SCO, QNX, y Microsoft Windows), para ser utilizado por aplicaciones de más alto nivel de gestión de centralitas [2].

Una de las aplicaciones que utilizan el API de EDIABAS se denomina ToolSet 32. Ésta aplicación muestra el catálogo de operaciones de cualquier centralita y permite ejecutar cualquiera de estas operaciones. Además, incluye un modo de depuración que graba en un fichero las tramas de bytes enviadas y recibidas con las centralitas. La figura 2.13 muestra la pantalla inicial de ToolSet 32.

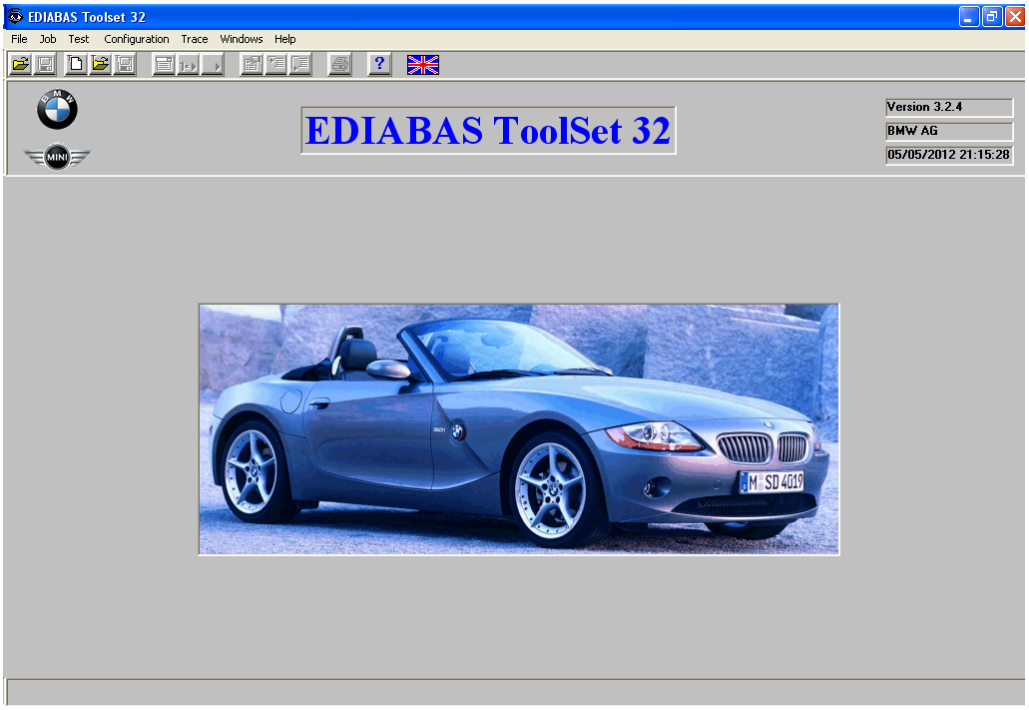


Figura 2.13: Pantalla de inicio de EDIABAS ToolSet 32.

2.4. Transmisión en serie

Se denomina transmisión en serie al envío y recepción de datos en una unidad de información cada vez, al contrario de lo que sería la transmisión en paralelo, donde se envían las unidades de información de manera simultánea.

La transmisión puede ser síncrona o asíncrona; la más común es la asíncrona, que se realiza mediante un aviso de comienzo (bit de *start*) y uno o varios avisos de de parada (bits de *stop*). Entre los bits de *start* y *stop* se envían "ceros" y "unos" lógicos, es decir, bits de información, con un mínimo de 4 y un máximo de 9. Éste bloque de bits de información se denomina palabra o carácter. La figura 2.14 refleja el envío de un carácter.

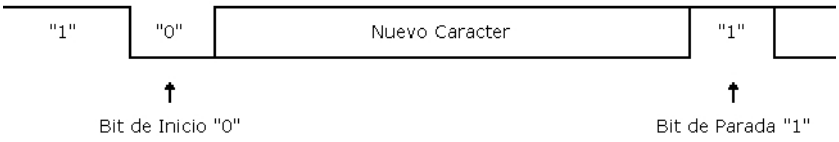


Figura 2.14: Transmisión en serie de un carácter.

La velocidad de la transmisión es expresada en bits por segundo o baudios por segundo. El receptor y el emisor deben tener un reloj interno que genere los baudios para poder transmitir a la misma velocidad. En el caso de la comunicación asíncrona, tanto el emisor como el receptor ponen en común sus relojes al comienzo del bit de start y mantienen la sincronía durante la transmisión del carácter.

Según el sentido de la transmisión y si se da al mismo tiempo entre los interlocutores, existen los siguientes tipos de transmisión en serie:

- **Simplex**: los datos viajan sólo en un sentido.
- **Half duplex**: los datos pueden viajar en los dos sentidos pero no al mismo tiempo.
- **Full duplex**: los datos pueden viajar en los dos sentidos y al mismo tiempo.

Respecto a las señales físicas, los bits se envían mediante señales eléctricas moduladas en amplitud. Existen varios estándares para representar los bits de información, los más conocidos son el RS-232C y el TTL. La tabla 2.2 representa los niveles de tensión para los bits según ambos estándares.

	Bit de "1" lógico	**Bit de "0" lógico**
TTL	+5V	0V
RS-232C	-3V a -15V	+3V a +15V

Cuadro 2.2: Niveles de tensión de los estándares TTL y RS232-C.

La transmisión serie también incorpora mecanismos opcionales de comproba-

ción de errores en la comunicación, uno de ellos es el bit de paridad que puede ser par o impar. Éste bit opcional se transmite al final del carácter y toma los siguientes valores en función de si la paridad es par o impar:

- **Paridad par**: el bit de paridad será cero, cuando el número de bit "unos" que contienen los datos a transmitir sea un número par, y el bit de paridad será "uno" cuando los datos que se mandan contienen un número impar de unos. La suma de los bits que son "unos", contando datos y bit de paridad dará siempre como resultado un número par de "unos".
- **Paridad impar**: en el sistema de paridad impar, el número de unos (datos más paridad) siempre debe ser impar.

Los circuitos que implementan los estándares de transmisión en serie son denominados UARTs o USARTS que son las siglas en inglés de *Universal Asynchronous/Synchronous Receiver/Transmitter* (Transmisor y Receptor Síncrono Asíncrono Universal). La función de las UARTs es transformar los datos en paralelo que llegan desde el bus de datos, a señales en serie que se envían a otro dispositivo a través de un interfaz, y viceversa.

2.4.1. Estándar RS-232C

El RS-232 (*Recommended Standard* 232) fue introducido en 1962 y posteriormente revisado por la EIA (Electronic Industries Association) en 1969. Ésta revisión, conocida como EIA RS-232C, ha sido la más utilizada desde entonces [11].

El estándar EIA RS-232C establece varios tipos de conectores, entre ellos el conector de tipo DB-9 mostrado en la figura 2.15 cuyas líneas de comunicación son las siguientes:

- **Pin 1 - CD**: *Carrier Detect*, detección de portadora.
- **Pin 2 - RXD**: *Receive Data*, recepción de datos.
- **Pin 3 - TXD**: *Transmit Data*, envío de datos.
- **Pin 4 - DTR**: *Data Terminal Ready*, terminal de datos preparado.
- **Pin 5 - GND**: *Ground*, tierra de señal.
- **Pin 6 - DSR**: *Data Set Ready*, dispositivo preparado.
- **Pin 7 - RTS**: *Request to Send*, petición de envío
- **Pin 8 - CTS**: *Clear to Send*, preparado para transmitir
- **Pin 9 - RI**: *Ring Indicator*, indicador de llamada entrante

La conexión mínima que se necesita para comunicar dos equipos en serie se hace conectando las siguientes líneas:

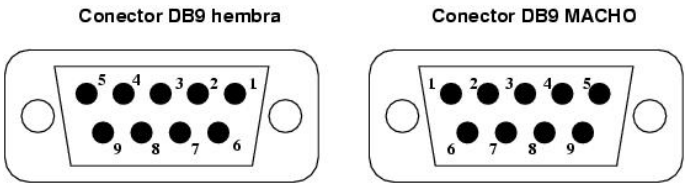


Figura 2.15: Conectores serie DB-9.

- Línea RX del equipo 1 con línea TX del equipo 2.
- Línea TX del equipo 1 con línea RX del equipo 2.
- Líneas GND conectadas entre el equipo 1 y el equipo 2.

Con ésta conexión mínima se pierden algunas de las funcionalidades del puerto serie pero se puede establecer una comunicación.

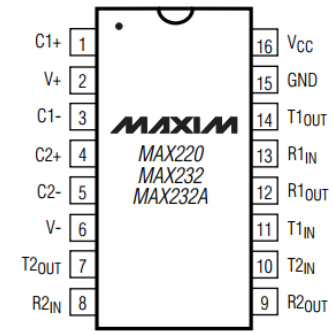
2.4.2. Conversor TTL a RS-232C

El MAX232 es un circuito integrado de la compañía Maxim que convierte las señales de un puerto serie RS-232C a señales compatibles con los niveles TTL de circuitos lógicos, por ejemplo en la conexión de un PC (con puerto serie RS-232C) a un microcontrolador (puertos serie con niveles TTL).

El MAX232 dispone internamente de 4 conversores de nivel TTL a RS-232C. Normalmente se utilizan sólo dos, para la conversión de las señales RX y TX de los dos equipos involucrados en la transmisión, aunque es frecuente utilizar los otros dos conversores para las señales CTS y RTS.

Para alcanzar los niveles de voltaje del RS-232C (aprox. $\pm$ 7.5 V) a partir de la alimentación de +5V, el MAX232 utiliza multiplicadores de voltaje internos y condensadores externos. Esto es de mucha utilidad para la implementación de puertos serie RS-232C en dispositivos que tengan una alimentación simple de +5 V [25].

En la figura 2.16 se puede observar la configuración de pines del integrado MAX232 en formato DIP/SO, un esquema del interior, y la tabla de valores de los condensadores según el tipo de integrado.



DIP/SO

CAPACITANCE (µF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	4.7	4.7	10	10	4.7
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

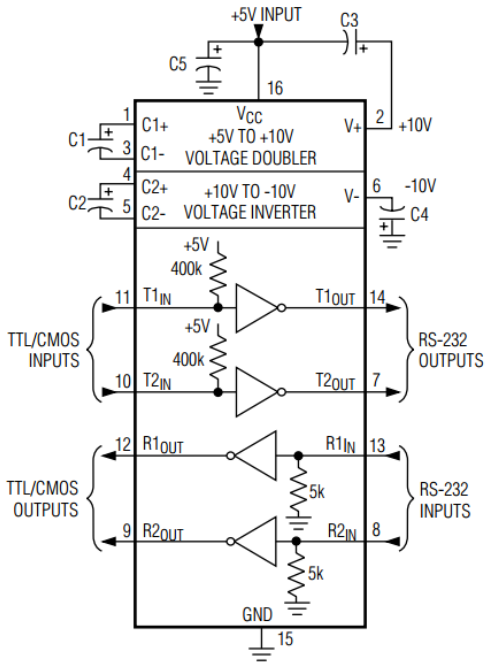


Figura 2.16: Integrado MAX232.

2.5. AVR

2.5.1. Introducción a los microcontroladores

Un microcontrolador (μC, UC o MCU), es un circuito integrado programable. Incluye en su propio encapsulado las tres unidades funcionales básicas de una computadora: unidad central de procesamiento, memoria y periféricos de entrada/salida.

Se diseñan para reducir el coste económico y energético de un sistema particular acotando las características del microcontrolador a las necesidades del sistema. Por ello, el tamaño de la unidad central de procesamiento, la cantidad de memoria y los periféricos incluidos dependerán del sistema donde se incluya.

La figura 2.17 muestra el esquema básico de un microcontrolador.

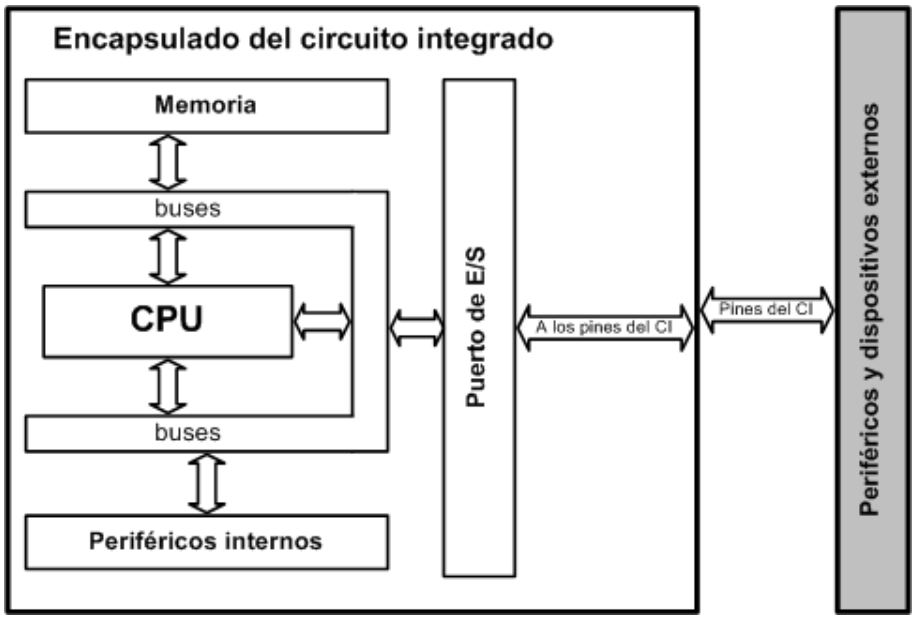


Figura 2.17: Estructura básica de un microcontrolador

Arquitectura

Los microcontroladores se basan generalmente en dos arquitecturas: Von Neumann y Harvard. Ambas se diferencian en la forma de conexión de la memoria al procesador y en los buses que cada una necesita:

1. **Arquitectura Von Neumann**: en los microcontroladores que utilizan ésta arquitectura, se utiliza el mismo bus de entrada/salida para acceder a los dos tipos de memoria existentes (memoria de datos y memoria de programas).
2. **Arquitectura Harvard**: en los microcontroladores que utilizan ésta arquitectura, se utilizan buses segregados, de modo que cada tipo de memoria

(memoria de datos y memoria de programas), tiene para si misma un bus de datos, uno de direcciones y uno de control.

Procesador

Los elementos más importantes que constituyen el procesador de un microcontrolador son los siguientes:

- **Banco de registros**: son un espacio de memoria muy reducido pero necesario para cualquier microprocesador, de aquí se toman los datos para varias operaciones que debe realizar el resto de los circuitos del procesador. Los registros sirven para almacenar los resultados de la ejecución de instrucciones, cargar datos desde la memoria externa o almacenarlos en ella. Dependiendo del tamaño de registro que utilice el procesador para realizar las operaciones anteriores, se puede decir que el procesador es de 4, 8, 16, 32 ó 64 bits.
- **Unidad de control**: esta unidad es de las más importantes en el procesador, en ella recae la lógica necesaria para la decodificación y ejecución de las instrucciones, el control de los registros, de la ALU (Unidad Aritmetico Lógica), de los buses y de otros elementos integrados en el procesador.
- **Unidad aritmético-lógica**: el procesador delega las operaciones aritméticas y lógicas (álgebra de Boole) a esta unidad.
- **Buses**: son el medio de comunicación que utilizan los diferentes componentes del procesador para intercambiar información entre sí, en el caso de los microcontroladores, no es común que los buses estén reflejados en el encapsulado del circuito, ya que estos se destinan básicamente a las E/S de propósito general y periféricos del sistema. Existen tres tipos de buses: de dirección, de datos y de control.
- **Conjunto de instrucciones**: define las operaciones básicas que puede realizar el procesador, que conjugadas y organizadas forman lo que conocemos como software.

Memoria

La memoria en los microcontroladores por regla general esta ubicada dentro del mismo encapsulado, dado que el objetivo fundamental es obtener la mayor integración posible. Esta memoria esta dividida en dos grandes bloques: memoria de datos y memoria de instrucciones.

1. **Memoria de datos**: se trata de una memoria RAM está destinada al almacenamiento de información temporal que será utilizada por el procesador para realizar cálculos u otro tipo de operaciones lógicas. En el espacio de direcciones utilizado por la memoria RAM se ubican además los registros de trabajo del procesador y los de configuración y trabajo de los distintos

periféricos del microcontrolador.

2. **Memoria de instrucciones**: es la memoria donde reside el código (software) a ejecutar por el microcontrolador. Las tecnologías que se han utilizado para este tipo de memoria son ROM, PROM, EPROM, EEPROM y FLASH.

Interrupciones

Las interrupciones son esencialmente llamadas a subrutina generadas por los dispositivos físicos, al contrario de las subrutinas normales de un programa en ejecución. Como el salto de subrutina no es parte del hilo o secuencia de ejecución programada, el controlador guarda el estado del procesador en la pila de memoria y entra a ejecutar un código especial llamado "manejador de interrupciones"que atiende al periférico específico que generó la interrupción. Al terminar la rutina, una instrucción especial le indica al procesador el fin de la atención de la interrupción. En ese momento el controlador restablece el estado anterior, y el programa que se estaba ejecutando antes de la interrupción sigue como si nada hubiese pasado.

Periféricos

- **Entradas y salidas de propósito general**: también conocidos como puertos de E/S, generalmente agrupadas en puertos de 8 bits de longitud, permiten leer datos del exterior o escribir en ellos desde el interior del microcontrolador, el destino habitual es el trabajo con dispositivos simples como relés, LED, o cualquier otra cosa que se le ocurra al programador.
  Algunos puertos de E/S tienen características especiales que le permiten manejar salidas con determinados requerimientos de corriente, o incorporan mecanismos especiales de interrupción para el procesador.

- **Temporizadores y contadores**: se emplean para controlar periodos de tiempo (temporizadores) y para llevar la cuenta de acontecimientos que suceden en el exterior (contadores).
  Para la medida de tiempos se carga un registro con el valor adecuado y a continuación dicho valor se va incrementando o decrementando al ritmo de los impulsos de reloj o algún múltiplo hasta que se desborde y llegue a 0, momento en el que se produce un aviso que puede llegar a generar o no una interrupción según se haya configurado.

- **Conversor analógico/digital**: como es muy frecuente el trabajo con señales analógicas, éstas deben ser convertidas a digital y por ello muchos microcontroladores incorporan un conversor analógico-digital, el cual se utiliza para tomar datos de varias entradas diferentes que se seleccionan mediante un multiplexor.

- **Comparadores**: son circuitos analógicos basados en amplificadores operacionales que tienen la característica de comparar dos señales analógicas y dar

como salida los niveles lógicos '0' o '1' en dependencia del resultado de la comparación.

- **PWM (Pulse-Width Modulation o Modulación por Ancho de Pulsos)**: es una técnica utilizada para usar una señal digital como si de una analógica se tratase. Es un periférico utilizado sobre todo para el control de motores.
- **Puertos de comunicación**: con objeto de permitir la interconexión del microcontrolador con otros periféricos externos bajo sus propias normas y protocolos, se dota al mismo de varios recursos que permitan esta tarea, entre los que destacan:
  - Puerto serie: este periférico está presente en casi cualquier microcontrolador, normalmente en forma de UART (Universal Asynchronous Receiver Transmitter) o USART (Universal Synchronous Asynchronous Receiver Transmitter) dependiendo de si permiten o no el modo sincrónico de comunicación. La forma más común de completar el puerto serie es para comunicarlo con una PC mediante el interfaz RS-232C, aunque este periférico también se puede utilizar para interconectar dispositivos mediante otros estándares de comunicación como el RS-485.
  - SPI (Serial Peripheral Interface) es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos mediante un flujo de bits serie regulado por un reloj.
  - I^2C (Inter-Integrated Circuit): es un estándar de bus de comunicaciones serie similar al SPI pero requiere menos señales de comunicación.
  - USB (Universal Serial Bus): se trata de un estándar de comunicaciones entre dispositivos generalmente utilizado para conectar dispositivos a un PC.
  - CAN (Controller Area Network): es un protocolo de comunicaciones basado en una topología bus para la transmisión de mensajes en entornos distribuidos. Es uno de los protocolos más utilizados para la interconexión de dispositivos en automóviles.
- **Memoria de datos no volátil**: muchos microcontroladores han incorporado este tipo de memoria como un periférico más, para el almacenamiento de datos de configuración o de los procesos que se controlan. Esta memoria es independiente de la memoria de datos tipo RAM o la memoria de programas, en la que se almacena el código del programa a ejecutar por el procesador del microcontrolador.

2.5.2. Microcontroladores AVR

Los AVR son una familia de microcontroladores RISC de 8 bits del fabricante estadounidense Atmel. La arquitectura de los AVR fue concebida por dos estudiantes en el Norwegian Institute of Technology, y posteriormente refinada y desarrollada en Atmel Norway, la empresa subsidiaria de Atmel, fundada por los dos arquitectos del chip. Cuenta con bastantes aficionados debido a su diseño simple y su facilidad de programación [23]. Se pueden dividir en los siguientes grupos :

- **ATxmega**: procesadores muy potentes con de 16 a 384 kB de memoria flash programable, encapsulados de 44, 64 y 100 pines, capacidad de DMA, eventos, criptografía y amplio conjunto de periféricos con conversores analógico/digitales.
- **ATmega**: microcontroladores AVR grandes con 4 a 256 kB de memoria flash programable, encapsulados de 28 a 100 pines, conjunto de instrucciones extendido (multiplicación y direccionamiento de programas mayores) y amplio conjunto de periféricos.
- **ATtiny**: pequeños microcontroladores AVR con 0,5 a 8 kB de memoria flash programable, encapsulados de 6 a 20 pines y un limitado set de periféricos.
- **AT90USB**: ATmega integrado con controlador USB.
- **AT90CAN**: ATmega con controlador de bus CAN.
- **Tipos especiales**: algunos modelos especiales, por ejemplo, para el control de los cargadores de baterías, pantallas LCD y los controles de los motores o la iluminación.
- **AT90S**: tipos obsoletos, los AVRs clásicos

El AVR es una CPU de arquitectura Harvard, su conjunto de instrucciones está implementado físicamente y disponible en el mercado en diferentes dispositivos, que comparten el mismo núcleo AVR pero tienen distintos periféricos y cantidades de RAM y ROM.

Estos microcontroladores están soportados por tarjetas de desarrollo de costo razonable, capaces de descargar el código al microcontrolador, y por una versión de las herramientas GNU. Esto último es posible por su uniformidad en el acceso al espacio de memoria, propiedad de la que carecen los procesadores de memoria segmentada o por bancos, como el PIC o el 8051 y sus derivados.

Desarrollo de código

El desarrollo de código fuente para los microcontroladores AVR puede realizarse en lenguaje ensamblador o en lenguaje C/C++. Para éste último, existen una serie de herramientas de código abierto que incluyen, entre otros, un compilador

denominado AVR GCC, librerías denominadas AVR C *Runtime Library* (avr-libc), e incluso una compilación de dichas herramientas para la plataforma Microsoft Windows, denominada WinAVR.

Además, la compañía Atmel proporciona su propio entorno de desarrollo (IDE) para los microcontroladores AVR. Se trata del Atmel AVR Studio, que actualmente está en la versión 6. Éste entorno de desarrollo es compatible con la compilación de herramientas libres WinAVR.

Programación del AVR

Los microcontroladores AVR poseen un interfaz de programación ISP (*In System Programming*), que posibilita la programación del micro sobre la placa en la que esté finalmente montado. El conector para éste interfaz y sus correspondientes pines se ven reflejados en la figura 2.18.

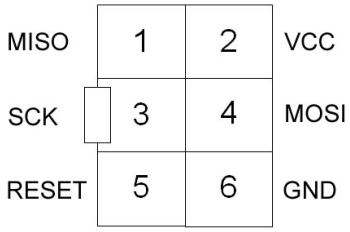


Figura 2.18: Pines del conector AVR ISP.

Para programar un microcontrolador a través de su interfaz ISP, se requiere un hardware adicional (programador) que sirva para transmitir el código compilado desde el PC al conector ISP. El programador utilizado en el presente proyecto es el AVRISP mkII (ver figura 2.19), que cuenta con un conector USB para la conexión con el PC, y un conector ISP para la placa donde esté integrado el microcontrolador.

Figura 2.19: Programador AVRISP mkII.

2.6. Bluetooth

Bluetooth es una especificación industrial para Redes Inalámbricas de Área Personal (WPAN) que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM[1] de los 2,4 GHz. Los principales objetivos que se pretenden conseguir con esta norma son [24]:

- Facilitar las comunicaciones entre equipos móviles y fijos.
- Eliminar cables y conectores entre éstos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales.

Figura 2.20: Logo del sistema Bluetooth.

2.6.1. Clases y versiones

Se denomina Bluetooth al protocolo de comunicaciones diseñado especialmente para dispositivos de bajo consumo, con una cobertura baja y basados en transceptores de bajo costo.

Gracias a este protocolo, los dispositivos que lo implementan pueden comunicarse entre ellos cuando se encuentran dentro de su alcance. Las comunicaciones se realizan por radiofrecuencia de forma que los dispositivos no tienen que estar alineados y pueden incluso estar en habitaciones separadas si la potencia de transmisión lo permite. Estos dispositivos se clasifican como "Clase 1", "Clase 2" o "Clase 3" en referencia a su potencia de transmisión, siendo totalmente compatibles los dispositivos de una clase con los de las otras. La tabla 2.3 muestra las distintas clases y sus características.

En la mayoría de los casos, la cobertura efectiva de un dispositivo de clase 2 se extiende cuando se conecta a un transceptor de clase 1. Esto es así gracias a la mayor sensibilidad y potencia de transmisión del dispositivo de clase 1, es decir, la mayor potencia de transmisión del dispositivo de clase 1 permite que la señal llegue con energía suficiente hasta el de clase 2. Por otra parte la mayor sensibilidad del dispositivo de clase 1 permite recibir la señal del otro pese a ser más débil.

[1]ISM (*Industrial, Scientific and Medical*) son bandas reservadas internacionalmente para uso no comercial de radiofrecuencia electromagnética en áreas industrial, científica y médica. En la actualidad estas bandas han sido popularizadas por su uso en comunicaciones WLAN (ej. Wi-Fi) o WPAN (ej. Bluetooth).

Clase	Potencia máxima permitida (mW)	Potencia máxima permitida (dBM)	Rango (aproximado)
Clase 1	100 mW	20 dBm	100m
Clase 2	2.5 mW	4 dBm	10m
Clase 3	1 mW	0 dBm	1m

Cuadro 2.3: Clases de Bluetooth.

Los dispositivos con Bluetooth también pueden clasificarse según su ancho de banda, como se puede observar en la tabla 2.4.

Versión	Ancho de banda
Versión 1.2	1 Mbit/s
Versión 2.0 + EDR	3 Mbit/s
Versión 3.0 +HS	24 Mbit/s
Versión 4.0	24 Mbit/s

Cuadro 2.4: Versiones de Bluetooth.

2.6.2. Arquitectura

La arquitectura de un sistema Bluetooth se basa en una pila de protocolos ordenados en capas, como muestra la figura 2.21.

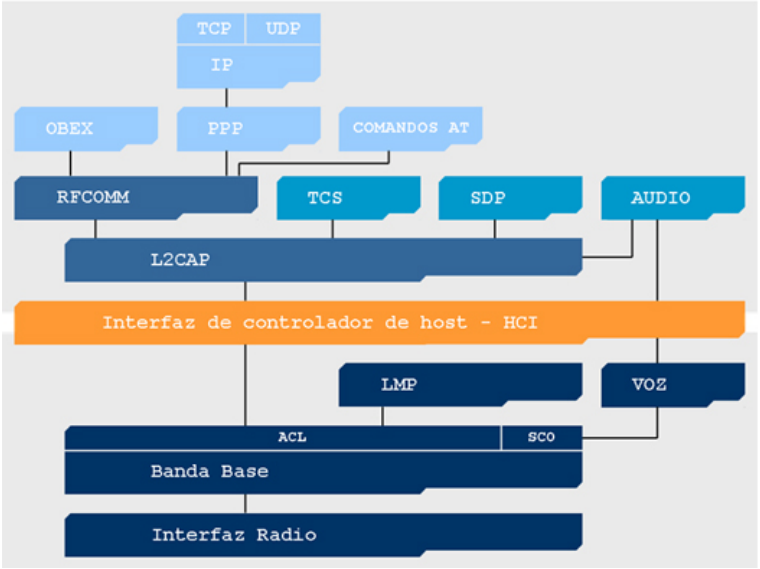


Figura 2.21: Pila de la arquitectura Bluetooth.

El interfaz controlador del host o HCI es la capa que separa la parte hardware de un sistema Bluetooth, de la parte software.El propio HCI está implementado parcialmente el el software y en el firmware del hardware [14]. La tabla 2.5 muestra la descripción de los protocolos que aparecen en la pila.

Protocolo	**Descripción**
TCP, UDP, IP, OBEX, PPP, COMANDOS AT	Pertenecen a la capa de aplicación. Componen los distintos perfiles de uso de los dispositivos Bluetooth, los cuales indican cómo las aplicaciones deben usar la pila de protocolos Bluetooth.
RFCOMM	Proporciona un interfaz que emula una línea RS-232C.
TCS (*Telephony Control System*)	Proporciona servicios de telefonía.
SDP (*Service Discovery Protocol*)	Se utiliza para encontrar otros dispositivos Bluetooth remotos.
L2CAP	Multiplexa los datos de las capas superiores y los convierte en paquetes de distinto tamaño según los requerimientos de las capas inferiores.
HCI	Gestiona las comunicaciones entre el hardware Bluetooth y el software.
LMP (*Link Manager Protocol*)	Controla y configura los enlaces a otros dispositivos Bluetooth.
ACL	Tipo de enlace a un dispositivo Bluetooth asíncrono, no orientado a conexión.
SCO	Tipo de enlace a un dispositivo Bluetooth síncrono orientado a conexión. Utilizado para transmisiones en tiempo real como las conexiones telefónicas.
Banda Base	Controla la capa física de enlace, los saltos de frecuencia y el agrupamiento de paquetes.
Radio	Modula y demodula los datos para la transmisión a través del aire.

Cuadro 2.5: Descripción de los protocolos de Bluetooth.

En el desarrollo de aplicaciones no es necesario conocer los detalles sobre cómo están implementadas las capas de la pila Bluetooth. Sin embargo, entender cómo trabaja la radio en las comunicaciones Bluetooth es importante. La radio en un dispositivo Bluetooth es la capa más baja de la pila de protocolos. Como se ha mencionado anteriormente, trabaja en la banda de los 2.4 GHz, compartiendo la misma banda con otras tecnologías como el Wi-Fi (IEEE 82.11b/g).

La radio Bluetooth utiliza una técnica de señales llamada espectro amplio de saltos de frecuencia o FHSS (*Frequency Hopping Spread Spectrum*). La banda de radio esta dividida en 79 subcanales. La radio Bluetooth usa uno de éstos canales de frecuencia al mismo tiempo. La radio salta de canal en canal permaneciendo 625 microsegundos en cada uno de ellos. Esto produce 1.600 saltos de frecuencia en cada segundo. Los saltos de frecuencia son utilizados para reducir las interferencias causadas por dispositivos Bluetooth cercanos u otros dispositivos que utilicen la misma banda de frecuencia.

2.6.3. Topologías de red

Hay varias formas en las que los dispositivos Bluetooth se pueden conectar entre sí. La forma más sencilla es el enlace punto a punto. Si más de dos dispositivos quieren establecer una conexión, se forma una *piconet*, que consta de un nodo maestro y hasta siete nodos esclavos a una distancia de 10 metros. Compartiendo el mismo espacio físico, pueden encontrarse varias *piconets*, y se pueden conectar mediante un nodo puente. Al conjunto de varias *piconets* interconectadas, se le denomina *scatternet* [21]. La figura 2.22 muestra varias *piconets* interconectadas formando una *scatternet*.

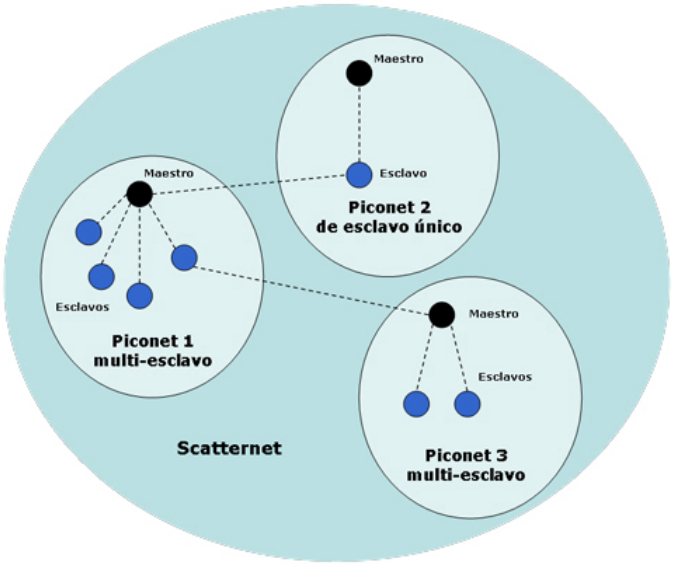


Figura 2.22: Topologías de red Bluetooth.

2.6.4. Establecimiento de la conexión

Todos los dispositivos Bluetooth permanecen en modo *standby* por defecto. En modo *standby*, los dispositivos no conectados, escuchan de manera periódica si hay mensajes de tipo *inquiry* o *page*, a ésta escucha se la denomina escaneo [19].

El escaneo esta dividido en dos tipos: *inquiry scan* y *page scan*. El primero de

ellos se produce cuando la unidad de Bluetooth comprueba si hay otras unidades cerca. Cualquier unidad cercana que este configurada como *discoverable*, responderá al *inquiry scan*. Es posible configurar una unidad de Bluetooth como *non discoverable*. El resultado del escaneo es la dirección Bluetooth de cada unidad, un identificador único de 48 bits (su MAC).

Además, cada unidad escaneada puede ser configurada como "conectable" o no "conectable". Si se configura como "conectable" responderá al *page scan*. Éste modo de escaneo proporciona información útil sobre como debe inicializarse la conexión.

2.6.5. Seguridad

La especificación Bluetooth define 3 niveles de seguridad: Nivel 1, canal no seguro; Nivel 2, seguridad a nivel de servicio; y Nivel 3, seguridad a nivel de enlace.

En el nivel no seguro, los dispositivos no ejecutan ningún tipo de acciones de seguridad. La conexión se establece y las transmisiones de datos tienen lugar. Para explicar el resto de niveles es necesario explicar algunas definiciones:

- **Autenticación**: verificar la identidad del dispositivo. Es usado para crear vínculos de confianza entre dispositivos.
- ***Paring***: autenticación y almacenamiento de una clave de enlace entre dos dispositivos. La clave de enlace es un código que se introduce manualmente en ambos lados.
- ***Bonding***: igual que el *paring*, pero las claves de enlace se mantienen en memoria para futuros usos.
- **Autorización**: concesión a un dispositivo remoto para acceder a un servicio.

Tanto el nivel 2 como el nivel 3 de seguridad, proporcionan autenticación y autorización, pero en distintas etapas del establecimiento de conexión. El tercer nivel de seguridad proporciona además servicios de cifrado, siendo el más seguro, pero menos flexible que el nivel 2.

Los mecanismos de seguridad de Bluetooth son suficientes para muchas aplicaciones, sin embargo, si éstos mecanismos embebidos no proporcionasen la seguridad necesaria, se pueden introducir esquemas de cifrado más fuertes en el nivel de capa de aplicación.

2.7. Android

2.7.1. Consideraciones generales

Android es un sistema operativo basado en Linux para dispositivos móviles como smartphones y tablets. Está desarrollado por la Open Handset Alliance, perteneciente a Google y a otras compañías [22].

Figura 2.23: Robot logotipo de Android.

Android es una plataforma de código abierto. Esto quiere decir, que cualquier desarrollador puede crear y desarrollar aplicaciones escritas con lenguaje Java, C u otros lenguajes y compilarlas a código nativo de ARM, que es la arquitectura que suelen utilizar los dispositivos físicos donde está Android instalado.

Además posee las siguientes características:

- ***Framework* de aplicaciones**: permite el reuso y el reemplazo de los componentes.
- **Máquina virtual Dalvik**: se trata de una máquina virtual optimizada para dispositivos móviles, es distinta de la máquina virtual java.
- **Navegador integrado**: cuenta con un navegador integrado en el S.O basado en el motor open source WebKit.
- **Gráficos optimizados**: posee soporte 2D gracias a la librería SGL y soporte 3D basados en la especificación OpenGL ES 1.X/2.0 (dependiendo de la versión de Android).
- **Almacenamiento de datos estructurados**: mediante las librerías SQLite.
- **Soporte multimedia de formatos comunes**: soporte para audio, vídeo e imágenes planas en soportes comunes como MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF, etc.
- **Telefonía GSM**: funciones de telefonía integradas y dependientes del dispositivo.
- **Otras comunicaciones inalámbricas**: tiene soporte para las tecnologías Bluetooth, EDGE, 3g y WiFi, dependiente del dispositivo.
- **Cámara, GPS, brújula y acelerómetro**: dependientes del dispositivo.

- **Entorno de desarrollo**: incluye un emulador, herramientas de depuración y un complemento para el IDE Eclipse.

2.7.2. Arquitectura

La figura 2.24 muestra los componentes del S.O Android. Cada sección se describe a continuación:

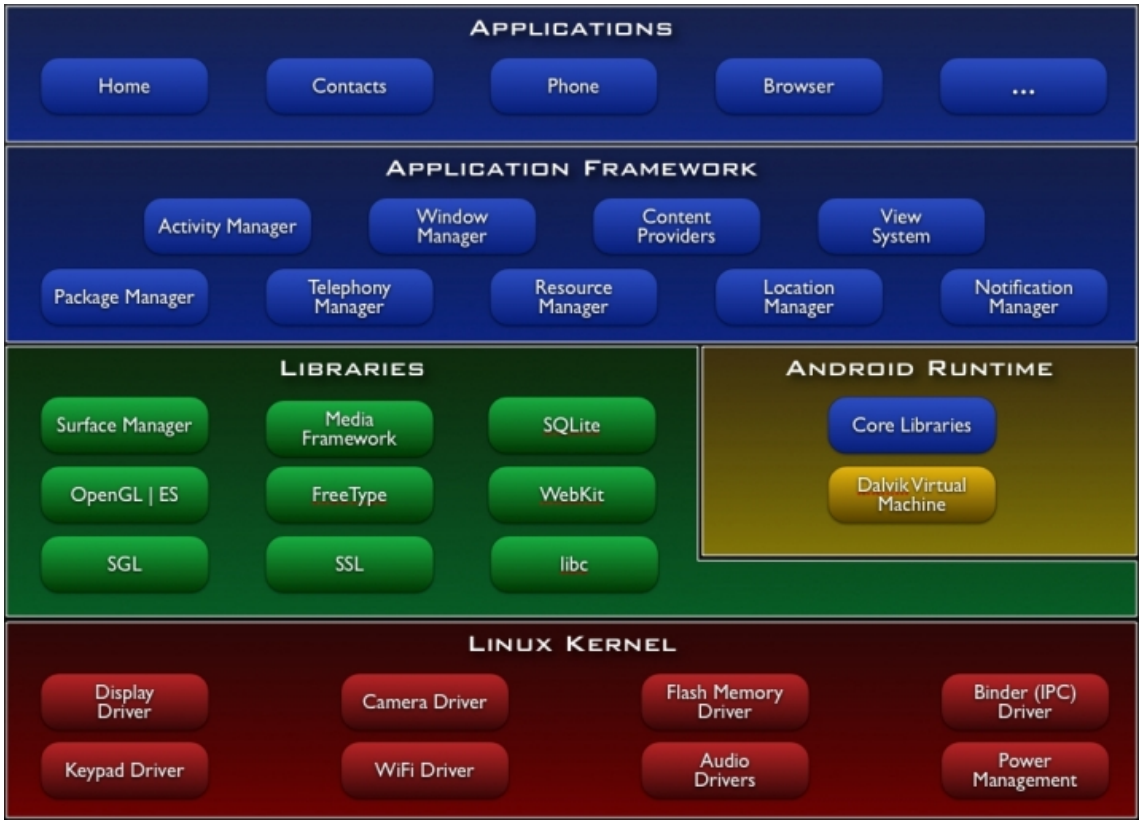


Figura 2.24: Arquitectura del S.O Android.

- **Aplicaciones**: las aplicaciones base incluyen un cliente de email, un programa de SMS, un calendario, mapas, navegador, contactos, y otros. Todas las aplicaciones están escritas en el lenguaje de programación Java.
- ***Framework* de aplicaciones**: los desarrolladores tienen acceso a los mismos APIs utilizados en las aplicaciones base (*core*). La arquitectura está diseñada para simplificar el reuso de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del *framework*). Éste mismo mecanismo permite que los componentes sean reemplazados por el usuario.
- **Librerías**: Android incluye un set de librerías C/C++ usadas por varios componentes del sistema. Estas capacidades se exponen a los desarrolladores a través del *framework* de aplicaciones de Android. Algunas son: System

C library (implementación de la librería C estándar), librerías de medios, librerías de gráficos, 3D, SQLite, entre otras.

- **Runtime de Android**: Android incluye un set de librerías base que proveen la mayor parte de las funcionalidades disponibles en las librerías base del lenguaje de programación Java. Cada aplicación Android corre su propio proceso, con su propia instancia de la Máquina Virtual Dalvik. Dalvik ha sido escrito de forma que un dispositivo puede correr múltiples máquinas virtuales de forma eficiente. Dalvik ejecuta archivos en el formato Dalvik ejecutable (.dex), el cual está optimizado para memoria mínima. La Máquina Virtual está basada en registros, y corre clases compiladas por el compilador de Java que han sido transformadas al formato .dex por la herramienta incluida "dx".
- **Núcleo - Linux**: Android depende de un Linux versión 2.6 para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red, y drivers de dispositivos. El núcleo también actúa como una capa de abstracción entre el hardware y el resto de la pila de software.

2.7.3. Aplicaciones

Las aplicaciones en Android se escriben habitualmente utilizando el lenguaje Java con el kit de desarrollo proporcionado por Google, el Android SDK (*Software Development Kit*). Éste kit contiene también un plugin para Eclipse denominado ADT (*Android Development Tools*), que extiende las capacidades del IDE para la creación de proyectos sobre Android.

El Android SDK compila el código en un *Android package*, es decir, un fichero con sufijo .apk que el sistema operativo utilizará para instalar la aplicación.

Una vez instalada una aplicación en un dispositivo, ésta se ejecuta en su propia *sandbox*, aislándola del resto de aplicaciones, mejorando la seguridad del conjunto.

Componentes

Las aplicaciones están construidas en bloques llamados componentes. Cada componente supone un punto de entrada a la aplicación; algunos son llamados desde otros componentes y otros son utilizables por el usuario. Hay cuatro tipos distintos:

1. ***Activities***: una *activity* representa una pantalla con un interfaz de usuario. Cada *Activity* se implementa como una subclase de la clase *Activity*.
2. ***Services***: un *service* es un componente que se ejecuta en segundo plano y que no posee una interfaz de usuario.
3. ***Content providers***: un *content provider* es un componente que gestiona información compartida. Se puede almacenar información en el dispositivo

móvil y que otras aplicaciones la utilicen utilizando los *content provider*.

4. ***Broadcast receivers***: un *broadcast receiver* es un componente que responde a anuncios realizados en modo *broadcast*. Las aplicaciones pueden a su vez inicializar anuncios de tipo *broadcast* para que los *broadcast receivers* los escuchen.

En el caso de los tres primeros componentes, se utiliza un objeto denominado *intent* para activarlos.

Intents

Un *intent* es una intención de realizar una acción en Android. Internamente, permite indicarle a Android que se desea hacer algo a través de otra actividad que se construyó específicamente para esa situación. De esta forma se posibilita la reutilización de componentes en las aplicaciones.

Los objetos de tipo intent envían mensajes asíncronos dentro de una aplicación o entre varias aplicaciones. Así, los Intents permiten enviar o recibir información desde y hacia otras actividades o servicios. También, permiten lanzar mensajes de tipo Broadcast para identificar cuando ciertos eventos han ocurrido.

Android soporta dos tipos de Intents: explícitos e implícitos. Los explícitos son aquellos que nombran el componente que se necesita, por ejemplo, la clase Java específica que se necesita para realizar alguna tarea. Por otro lado, los implícitos, indican el tipo de acción que se desea ejecutar y deja al S.O que busque la aplicación por defecto del sistema para realizar dicha acción.

Fichero *Manifest*

Toda aplicación Android necesita un archivo llamado "AndroidManifest.xml" en el directorio raíz del proyecto. El *Android Manifest* contiene información esencial necesaria sobre la aplicación de Android, información que además es requerida antes de poder ejecutar cualquier línea de código.

El *Android Manifest* contiene al menos los siguientes elementos:

- **Nombre del paquete Java**: éste nombre sirve como un identificador único de la aplicación.
- **Descripción de cada componente**: debe contener una descripción de cada componente que integra la aplicación, así como el nombre de la clase que lo implementa. De esta forma el sistema Android sabe que componentes existen en la aplicación y bajo que condiciones se ejecutarán.
- **Permisos**: en este elemento se determinan los permisos que tiene la aplicación para acceder a partes protegidas del API e interactuar con otras aplicaciones.

- **Versión del API**: es necesario declarar el nivel mínimo del API Android.
- **Librerías extras**: son aquellas librerías extras que la aplicación necesita, como por ejemplo la librería Google Maps.

Ciclo de vida de las *Activities*

Como se ha comentado anteriormente en la descripción de los componentes, las *Activities* son el componente que provee una pantalla gráfica al usuario para que interactúe con la aplicación. Una aplicación está compuesta de una o varias *activities* que son llamadas entre ellas. Cada *activity* tiene un ciclo de vida en el sistema donde cada estado depende del estado del resto de *activities* que se ejecutan en el dispositivo.

Hay que diferenciar entre *activity* y aplicación. Cuando una *activity* finaliza su ciclo de vida, la aplicación no tiene por qué terminar, salvo que sea la *activity* principal.

Se pueden establecer tres fases para una *activity*:

- **Activa o en ejecución**: se da cuando la *activity* se encuentra en primer plano y tiene el foco de atención (el usuario visualiza la *activity* en ese momento y puede interactuar con ella).
- **En pausa**: se da cuando la *activity* pierde el foco de atención por existir otras *activities* superpuestas con las que el usuario está interactuando, pero sigue siendo visible para el usuario si vuelve a ella.
- **Detenida**: se da cuando la *activity* ha sido ocultada en tu totalidad por otra.

La figura 2.25 muestra los distintos estados de una *activity* en Android. Los óvalos representan los estados más importantes en los que puede estar una *activity*, mientras que los rectángulos representan las funciones que son llamadas cuando la *activity* transita entre los estados.

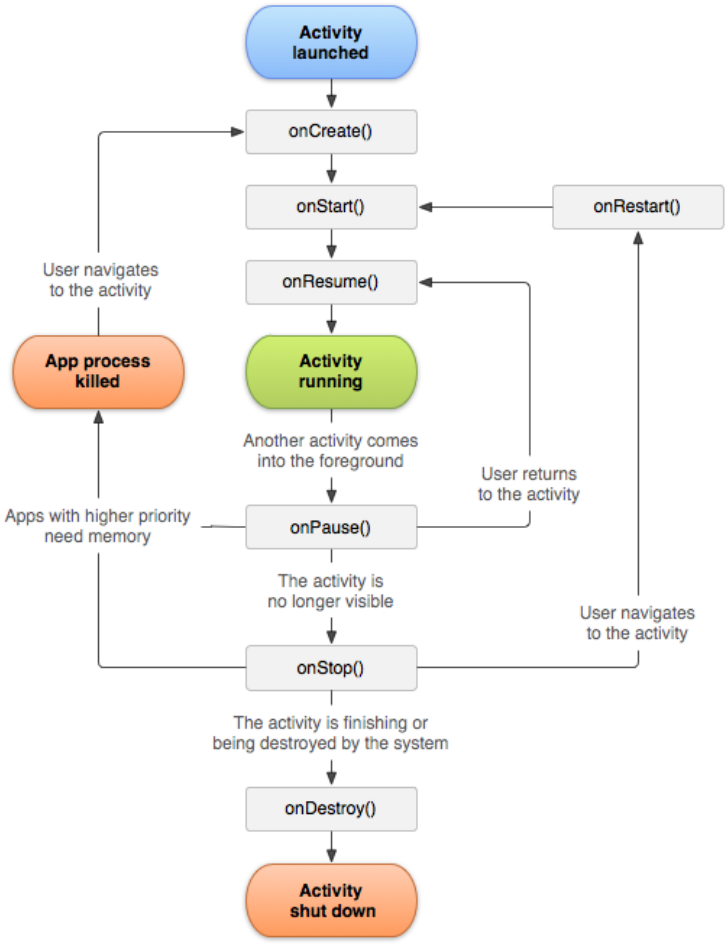


Figura 2.25: Ciclo de vida de una *activity* en Android.

De los estados que aparecen en la figura se pueden observar tres bucles de ejecución:

1. **Ciclo de vida completo**: sucede entre las funciones *onCreate* y *onDestroy*. Todas las *activities* deben implementar *onCreate*, utilizado para aplicar la configuración inicial de una nueva instancia de la *activity*. Por otro lado *onDestroy* se utiliza para liberar los recursos que quedan en la *activity*.

2. **Tiempo de vida visible**: sucede entre las funciones *onStart* y *onResume*. En éste tiempo el usuario puede ver la *activity* en la pantalla aunque no esté interactuando con ella (hay otra *activity* en primer plano).

3. **Tiempo de vida en primer plano**: sucede entre las funciones *onResume* y *onPause*. En éste periodo la *activity* está en primer plano y el usuario puede interactuar con ella.

Cuando una *activity* está en pausa o detenida, el sistema puede terminar su ejecución, ya sea porque le pide finalizar o simplemente porque elimina su proceso. Para salvaguardar su información del estado, Android ofrece métodos en su API

que pueden ser utilizados antes de que la *activity* sea eliminada. De esta forma, cuando vuelve a tener el foco de atención, puede recuperar el último estado en que estuvo.

Capítulo 3

Análisis del sistema

En éste capítulo se va a realizar un análisis del sistema que se desea construir, que contará tanto con una parte de hardware como con una de software. El análisis consistirá en una toma de requisitos y una descripción mediante casos de uso.

3.1. Requisitos de usuario

Los requisitos de usuario se dividen en requisitos de capacidad y requisitos de restricción. La información que va a contener cada requisito es la siguiente:

- **Identificador**: se trata de un identificador único para cada requisito. Los requisitos de capacidad tendrán un identificador con la forma RUC-XX, mientras que los requisitos de restricción tendrán un identificador con la forma RUR-XX. En ambos casos, XX representa un número decimal que sirve para enumerar de forma secuencial los requisitos.
- **Nombre**: define un nombre para el requisito.
- **Descripción**: se realiza una descripción detallada del requisito.
- **Necesidad**: establece el grado de necesidad de que el requisito esté presente en el desarrollo. Su valor está dentro del rango de 1 a 4, siendo uno la necesidad más baja y 4 la necesidad más alta.
- **Prioridad**: establece el grado de prioridad del requisito en el proyecto. Su valor está dentro del rango de 1 a 4, siendo uno la prioridad más baja y 4 la prioridad más alta.
- **Estabilidad**: establece la sensibilidad del requisito a ser modificado a lo largo del proyecto. Su valor puede ser “Estable” o “No Estable”.

3.1.1. Requisitos de capacidad

A continuación se listan los requisitos de capacidad:

Identificador	RUC-01
Nombre	Abrir ventanillas.
Descripción	El usuario será capaz de abrir las ventanillas.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Cuadro 3.1: Requisito de capacidad número 1.

Identificador	RUC-02
Nombre	Cerrar ventanillas
Descripción	El usuario será capaz de cerrar las ventanillas.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Cuadro 3.2: Requisito de capacidad número 2.

Identificador	RUC-03
Nombre	Abrir techo.
Descripción	El usuario será capaz de abrir el techo eléctrico.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Cuadro 3.3: Requisito de capacidad número 3.

Identificador	RUC-04
Nombre	Cerrar techo.
Descripción	El usuario será capaz de cerrar el techo eléctrico
Necesidad	4
Prioridad	4
Estabilidad	Estable

Cuadro 3.4: Requisito de capacidad número 4.

Identificador	RUC-05
Nombre	Abrir vehículo.
Descripción	El usuario será capaz de abrir el vehículo
Necesidad	4
Prioridad	4
Estabilidad	Estable

Cuadro 3.5: Requisito de capacidad número 5.

Identificador	RUC-06
Nombre	Cerrar vehículo.
Descripción	El usuario será capaz de cerrar el vehículo
Necesidad	4
Prioridad	4
Estabilidad	Estable

Cuadro 3.6: Requisito de capacidad número 6.

Identificador	RUC-07
Nombre	Cierre seguro vehículo.
Descripción	El usuario será capaz de cerrar el vehículo con doble cierre
Necesidad	4
Prioridad	4
Estabilidad	Estable

Cuadro 3.7: Requisito de capacidad número 7.

Identificador	RUC-08
Nombre	Activar limpiaparabrisas.
Descripción	El usuario será capaz de activar el limpiaparabrisas
Necesidad	4
Prioridad	4
Estabilidad	Estable

Cuadro 3.8: Requisito de capacidad número 8.

Identificador	RUC-09
Nombre	Reconocimiento de voz.
Descripción	La aplicación será capaz de reconocer comandos de voz del usuario.
Necesidad	2
Prioridad	2
Estabilidad	Estable

Cuadro 3.9: Requisito de capacidad número 9.

Identificador	RUC-10
Nombre	Ejecución a través de pantalla táctil.
Descripción	El usuario será capaz de ejecutar las acciones de la aplicación desde la pantalla táctil
Necesidad	4
Prioridad	4
Estabilidad	Estable

Cuadro 3.10: Requisito de capacidad número 10.

Identificador	RUC-11
Nombre	Mostrar texto al usuario.
Descripción	La aplicación mostrará un texto sobre las acciones que vaya a ejecutar.
Necesidad	4
Prioridad	2
Estabilidad	Estable

Cuadro 3.11: Requisito de capacidad número 11.

Identificador	RUC-12
Nombre	Expresión por voz.
Descripción	La aplicación expresará mediante una voz las acciones que vaya a ejecutar.
Necesidad	2
Prioridad	2
Estabilidad	Estable

Cuadro 3.12: Requisito de capacidad número 12.

Identificador	RUC-13
Nombre	Informar de error.
Descripción	La aplicación avisará al usuario sobre los errores que se produzcan.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Cuadro 3.13: Requisito de capacidad número 13.

3.1.2. Requisitos de restricción

A continuación se listan los requisitos de restricción:

Identificador	RUR-01
Nombre	Versión de Android.
Descripción	La aplicación funcionará en dispositivos móviles con la versión 2.3 de Android o superior.
Necesidad	4
Prioridad	4
Estabilidad	No Estable

Cuadro 3.14: Requisito de restricción número 1.

Identificador	RUR-02
Nombre	Conexión a internet para reconocimiento de voz.
Descripción	La aplicación necesitará conexión con internet para ejecutar el reconocimiento de voz.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Cuadro 3.15: Requisito de restricción número 2.

Identificador	RUR-03
Nombre	Idioma de la aplicación.
Descripción	La aplicación utilizará el idioma español.
Necesidad	3
Prioridad	3
Estabilidad	No Estable

Cuadro 3.16: Requisito de restricción número 3.

Identificador	RUR-04
Nombre	Conexión al vehículo.
Descripción	La aplicación necesitará conexión con el vehículo vía Bluetooth.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Cuadro 3.17: Requisito de restricción número 4.

Identificador	RUR-05
Nombre	No bloquear la pantalla.
Descripción	La aplicación deberá mantener la pantalla desbloqueada para que el usuario pueda utilizar la función de reconocimiento de voz mientras conduce, reduciendo el tiempo de distracción al tener que pulsar sólo un botón.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Cuadro 3.18: Requisito de restricción número 5.

3.2. Casos de uso

En este apartado se muestran los casos de uso del sistema. Cada uno de ellos consta de un diagrama que representa el caso de uso y de una especificación con los siguientes campos:

- **Identificador**: se trata de un identificador único para cada caso de uso. El identificador tiene la forma CU-XX, donde XX representa un número decimal que sirve para enumerar de forma secuencial los casos de uso.
- **Nombre**: define un nombre para el caso de uso.
- **Descripción**: se realiza una descripción detallada del caso de uso.
- **Pre-condiciones**: establece las condiciones que deben darse para que se realice el caso de uso.
- **Post-condiciones**: establece las condiciones que se dan al ejecutarse el caso de uso.

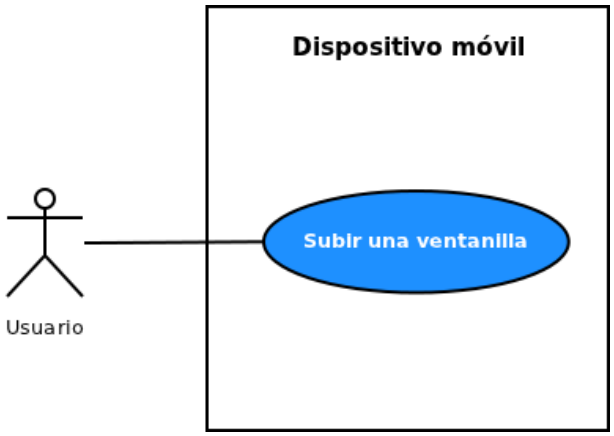


Figura 3.1: Caso de uso número 1.

Identificador	CU-01
Nombre	Subir ventanilla.
Descripción	▪ El usuario inicia la aplicación. ▪ El usuario pulsa sobre el botón de reconocimiento de voz y dicta la orden de subir una de las ventanillas o bien accede al menú de acciones y selecciona dicha operación desde el menú. ▪ El móvil se comunica con el interfaz remoto conectado al vehículo transmitiendo la acción vía Bluetooth. ▪ El vehículo sube la ventanilla que había seleccionado el usuario.
Pre-condiciones	▪ La ventanilla seleccionada por el usuario no debe estar subida del todo. ▪ El móvil debe tener la conexión a internet activa si utiliza el reconocimiento de voz.
Post-condiciones	La ventanilla seleccionada por el usuario sube aproximadamente 1/3 o menos si llega antes al final del recorrido de subida.

Cuadro 3.19: Caso de uso número 1.

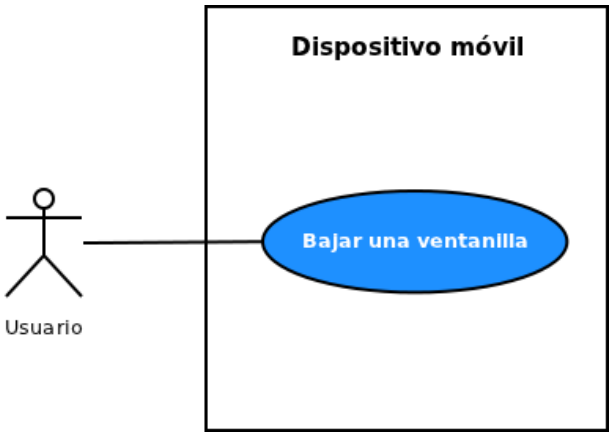


Figura 3.2: Caso de uso número 2.

Identificador	CU-02
Nombre	Bajar ventanilla.
Descripción	▪ El usuario inicia la aplicación. ▪ El usuario pulsa sobre el botón de reconocimiento de voz y dicta la orden de bajar una de las ventanillas o bien accede al menú de acciones y selecciona dicha operación desde el menú. ▪ El móvil se comunica con el interfaz remoto conectado al vehículo transmitiendo la acción vía Bluetooth. ▪ El vehículo baja la ventanilla que había seleccionado el usuario.
Pre-condiciones	▪ La ventanilla seleccionada por el usuario no debe estar bajada del todo. ▪ El móvil debe tener la conexión a internet activa si utiliza el reconocimiento de voz.
Post-condiciones	La ventanilla seleccionada por el usuario baja aproximadamente 1/3 o menos si llega antes al final del recorrido de bajada.

Cuadro 3.20: Caso de uso número 2.

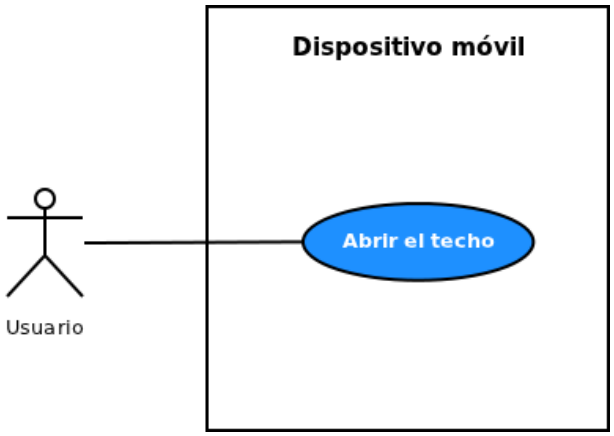


Figura 3.3: Caso de uso número 3.

Identificador	CU-03
Nombre	Abrir techo.
Descripción	▪ El usuario inicia la aplicación. ▪ El usuario pulsa sobre el botón de reconocimiento de voz y dicta la orden de bajar abrir el techo eléctrico o bien accede al menú de acciones y selecciona dicha operación desde el menú. ▪ El móvil se comunica con el interfaz remoto conectado al vehículo transmitiendo la acción vía Bluetooth. ▪ El vehículo abre el techo eléctrico.
Pre-condiciones	▪ El techo eléctrico no debe estar abierto del todo. ▪ El móvil debe tener la conexión a internet activa si utiliza el reconocimiento de voz.
Post-condiciones	El techo eléctrico se abre aproximadamente 1/3 o menos si llega antes al final del recorrido de apertura.

Cuadro 3.21: Caso de uso número 3.

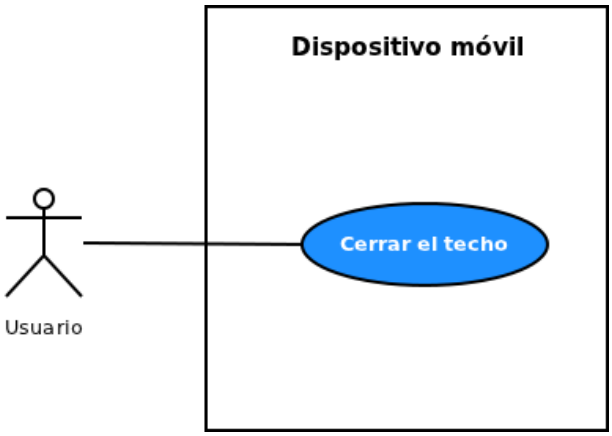


Figura 3.4: Caso de uso número 4.

Identificador	CU-04
Nombre	Cerrar techo.
Descripción	▪ El usuario inicia la aplicación. ▪ El usuario pulsa sobre el botón de reconocimiento de voz y dicta la orden de bajar cerrar el techo eléctrico o bien accede al menú de acciones y selecciona dicha operación desde el menú. ▪ El móvil se comunica con el interfaz remoto conectado al vehículo transmitiendo la acción vía Bluetooth. ▪ El vehículo cierra el techo eléctrico.
Pre-condiciones	▪ El techo eléctrico no debe estar cerrado del todo. ▪ El móvil debe tener la conexión a internet activa si utiliza el reconocimiento de voz.
Post-condiciones	El techo eléctrico se cierra aproximadamente 1/3 o menos si llega antes al final del recorrido de cierre.

Cuadro 3.22: Caso de uso número 4.

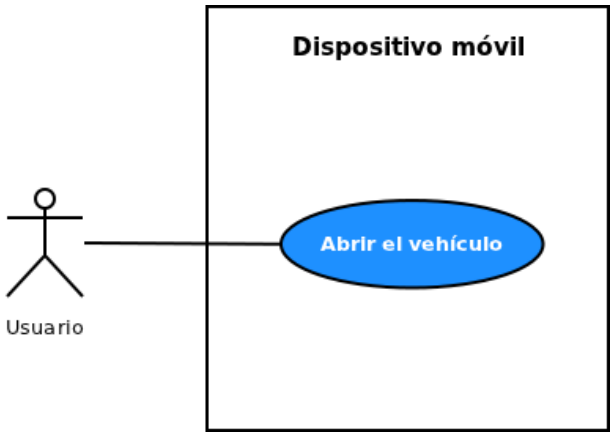


Figura 3.5: Caso de uso número 5.

Identificador	CU-05
Nombre	Abrir vehículo.
Descripción	▪ El usuario inicia la aplicación. ▪ El usuario pulsa sobre el botón de reconocimiento de voz y dicta la orden de abrir el vehículo o bien accede al menu de acciones y selecciona dicha operación desde el menú. ▪ El móvil se comunica con el interfaz remoto conectado al vehículo transmitiendo la acción vía Bluetooth. ▪ El vehículo abre todas las puertas y el maletero.
Pre-condiciones	▪ El vehículo debe tener al menos una puerta o el maletero cerrado. ▪ El móvil debe tener la conexión a internet activa si utiliza el reconocimiento de voz.
Post-condiciones	El vehículo abre todas sus puertas y el maletero.

Cuadro 3.23: Caso de uso número 5.

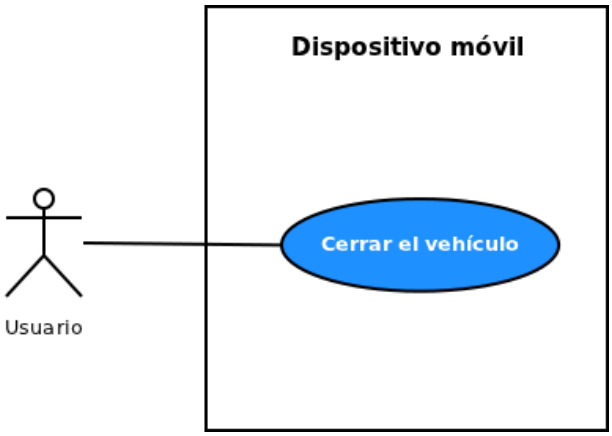


Figura 3.6: Caso de uso número 6.

Identificador	CU-06
Nombre	Cerrar vehículo.
Descripción	▪ El usuario inicia la aplicación. ▪ El usuario pulsa sobre el botón de reconocimiento de voz y dicta la orden de cerrar el vehículo o bien accede al menu de acciones y selecciona dicha operación desde el menú. ▪ El móvil se comunica con el interfaz remoto conectado al vehículo transmitiendo la acción vía Bluetooth. ▪ El vehículo activa el cierre centralizado.
Pre-condiciones	▪ El vehículo debe tener al menos una puerta o el maletero abierto. ▪ El móvil debe tener la conexión a internet activa si utiliza el reconocimiento de voz.
Post-condiciones	El vehículo ciera todas sus puertas y el maletero.

Cuadro 3.24: Caso de uso número 6.

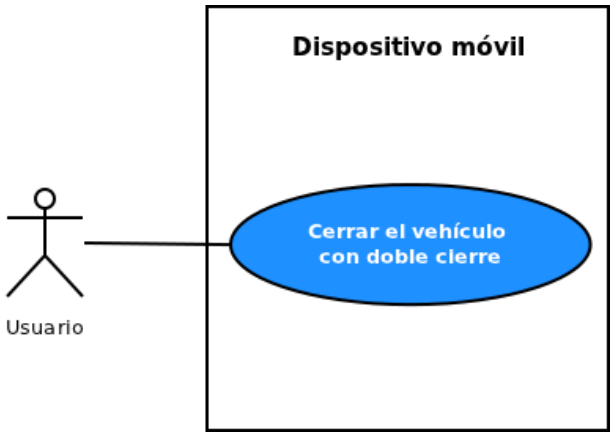


Figura 3.7: Caso de uso número 7.

Identificador	CU-07
Nombre	Cerrar vehículo con doble cierre.
Descripción	▪ El usuario inicia la aplicación. ▪ El usuario pulsa sobre el botón de reconocimiento de voz y dicta la orden de cerrar el vehículo con doble cierre o bien accede al menú de acciones y selecciona dicha operación desde el menú. ▪ El móvil se comunica con el interfaz remoto conectado al vehículo transmitiendo la acción vía Bluetooth. ▪ El vehículo activa el cierre centralizado con doble cierre.
Pre-condiciones	▪ El vehículo debe tener al menos una puerta o el maletero abierto. ▪ El móvil debe tener la conexión a internet activa si utiliza el reconocimiento de voz.
Post-condiciones	El vehículo cierra todas sus puertas y el maletero con doble cierre.

Cuadro 3.25: Caso de uso número 7.

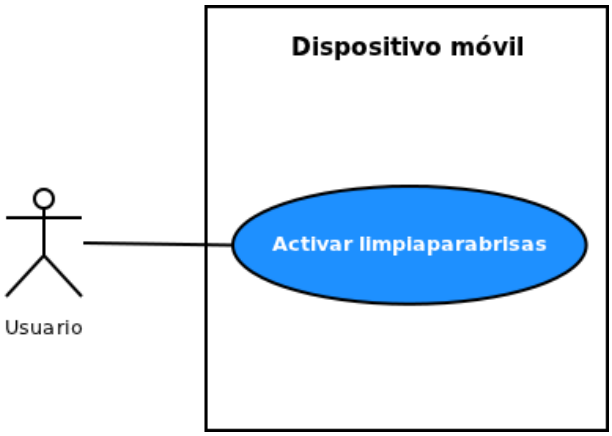


Figura 3.8: Caso de uso número 8.

Identificador	CU-08
Nombre	Limpiaparabrisas.
Descripción	▪ El usuario inicia la aplicación. ▪ El usuario pulsa sobre el botón de reconocimiento de voz y dicta la orden de activar el limpiaparabrisas o bien accede al menú de acciones y selecciona dicha operación desde el menú. ▪ El móvil se comunica con el interfaz remoto conectado al vehículo transmitiendo la acción vía Bluetooth. ▪ El vehículo activa el limpiaparabrisas una vez.
Pre-condiciones	▪ El móvil debe tener la conexión a internet activa si utiliza el reconocimiento de voz.
Post-condiciones	El vehículo activa el limpiaparabrisas una vez.

Cuadro 3.26: Caso de uso número 8.

Capítulo 4

Estudio de la comunicación con el vehículo

4.1. Objetivos

En este capítulo se detallan los pasos seguidos para poder realizar una comunicación con las centralitas del vehículo y en especial con la centralita ZKE (ver 2.3.2 en página 15), encargada de la gestión de los elevalunas, cierres, y el techo eléctrico entre otros elementos. Para alcanzar este objetivo, se utilizará una metodología basada en la ingeniería inversa sobre dos sistemas funcionales que realizan dicha comunicación: Carsoft, de Carsoft International y EDIABAS de BMW AG.

En las secciones posteriores del capítulo se ira detallando la consecución de los siguientes hitos:

- Reproducir la comunicación con la centralita ZKE en una mesa de trabajo.
- Analizar las comunicaciones mediante los sistemas Carsoft y Ediabas.
- Creación de un diccionario de funciones y códigos de los elementos del vehículo que gestiona la centralita ZKE.

4.2. Reproducción de la comunicación con la centralita ZKE en una mesa de trabajo

Para no dañar los sistemas eléctricos del vehículo en la realización de las pruebas (por ejemplo, mediante un cortocircuito), se ha preferido utilizar una reproducción sobre una mesa de trabajo de la comunicación con la centralita ZKE. Los elementos necesarios son: una centralita del mismo modelo, y un sistema de diagnosis (basado en un software y un interfaz hardware).

4.2.1. Centralita ZKE

En los vehículos BMW E34 como el del presente proyecto, existen dos modelos principales de centralitas ZKE [3]:

1. Modelos hasta 9/1991 con números de serie 61 35 8 356 09 en adelante.
2. Modelos desde 9/1991 con números de serie 61 35 8 355 812 en adelante.

La centralita ZKE utilizada se ha obtenido de un desguace, está localizada debajo del asiento trasero en el lado del conductor (ver figura 2.9 en página 16) y su número de serie es 61.35-8 355 818, correspondiente a un vehículo BMW E34 fabricado a partir del 9/91, compatible con el vehículo de pruebas.

Una vez obtenida la centralita, es necesario estudiar la función de cada conector y sus pines. Para ello se han utilizado como fuentes los documentos Bentley Manual [18] y el ETM [7] que incluyen los esquemas eléctricos de todo el vehículo. De las fuentes consultadas se han hallado entre otras, las siguientes conexiones de la centralita ZKE:

- Alimentación (+12V).
- Alimentación (GND).
- Líneas de datos.
- Conexiones a una centralita de relés (encargada de los motores de las ventanas, los cierres, el limpiaparabrisas,...).
- Conexiones a sensores de los elementos que controla la centralita.

Para poner en funcionamiento la centralita sobre la mesa de trabajo y testear su funcionamiento, se han utilizado el mínimo número de conexiones: las conexiones de alimentación y las conexiones de las líneas de datos (RX y TX).

Los conectores y los pines utilizados han sido los siguientes:

- **Conector X253, 26 pines, verde**:
  - Pin nº12 (entrada y salida): línea bi-direccional de datos TX. Cable de color blanco/violeta.

  - Pin nº24 (entrada): línea unidireccional de datos RX. Cable de color blanco/amarillo.

- **Conector X255, 26 pines amarillo**:
  - Pin nº9 (entrada): alimentación +12 V. Cable de color verde/rojo.
  - Pin nº10 (entrada): alimentación +12 V. Cable de color morado/negro.
- **Conector X332, 5 pines blanco**:
  - Pin nº1 (entrada): alimentación +12 V. Cable de color rojo/amarillo
  - Pin nº2 (tierra): cable de color marrón/naranja
  - Pin nº4 (entrada): alimentación +12 V. Cable de color rojo/marrón
  - Pin nº5 (entrada): alimentación +12 V. Cable de color rojo/violeta

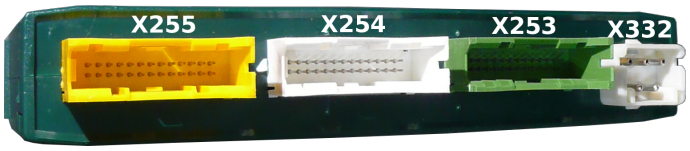


Figura 4.1: Conectores de la centralita ZKE.

4.2.2. Diagnosis

Para comprobar el correcto funcionamiento de la centralita obtenida del desguace, además de realizar las conexiones anteriores, es preciso realizar un diagnóstico y ver si se produce la comunicación con éxito. La figura 4.2 muestra el proceso a realizar en la diagnosis: un pc con el software de Carsoft instalado, conectado al interfaz hardware y éste último directamente a la centralita.

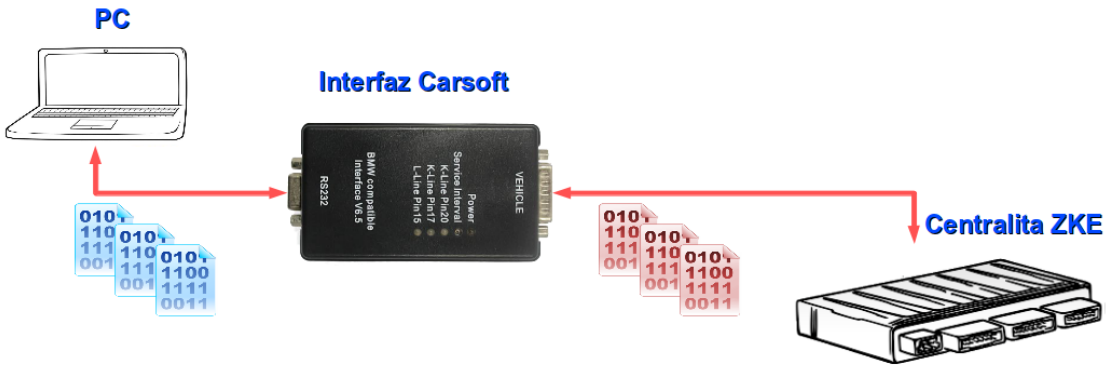


Figura 4.2: Conexión de Carsoft con la centralita ZKE

El interfaz hardware se conecta por un lado al PC y por otro lado al vehículo mediante conectores DB-9 y DB-15 respectivamente. El cable de conexión al PC

es un cable serie con conectores DB-9 en ambos extremos; por otro lado, el cable del interfaz al vehículo tiene un conector DB-15 en un extremo (el que se conecta al interfaz) y en el otro dos posibles conectores: un conector OBD de 20 pines especial para BMW, o bien un conector OBD II estándar. El vehículo de pruebas BMW E34 utiliza el conector OBD de 20 pines. En la figura 4.3 aparece el tipo de cable utilizado.

Figura 4.3: Cable OBD de 20 pines para el interfaz hardware de Carsoft

Para la conexión del interfaz hardware de Carsoft al circuito donde reside la centralita en la mesa de trabajo se ha utilizado un cable especial construido para la ocasión. El cable se ha conectado por un lado al conector DB-15 del interfaz hardware y por otro a las líneas de datos TX y RX y de alimentación del circuito. Para conocer la función de cada pin del conector DB-15, se ha utilizado un multímetro y el cable con el conector OBD de 20 pines mencionado anteriormente. El multímetro se ha puesto en modo medición de continuidad y se ha obtenido la correspondencia entre los pines TX (L), RX(K), +12V y GND del conector OBD I (ver 2.1 en página 15) y los del conector DB-15 del interfaz hardware de Carsoft. El cuadro 4.1 muestra la correspondencia.

Pines del conector DB-15	**Pines del conector OBD-20**	**Función**
2	20	Línea TX (K)
9	19	GND
15	14	+12V
4	15	Línea RX (L)
3	17	Vacía
1	7	Reseteo de intervalos de servicio

Cuadro 4.1: Interfaz hardware de Carsoft (DB-15) a OBD de 20 pines

Una vez realizadas las conexiones según el esquema de la figura 4.4, se ha

procedido mediante el software Carsoft a realizar un diagnóstico de la centralita ZKE mediante los siguientes pasos:

- Se abre el programa y a través del menú *Perform Diagnosis* se selecciona ZKE/ZVM como muestra la figura 4.5.
- Carsoft intenta identificar el modelo de centralita ZKE conectada al sistema como muestra la figura 4.6.
- Una vez que identifica la centralita, establece la comunicación con ella para realizar la diagnosis como muestra la figura 4.7.
- La centralita responde a la petición de diagnosis mostrando su estado y los errores acumulados en su memoria, como muestra la figura 4.8.

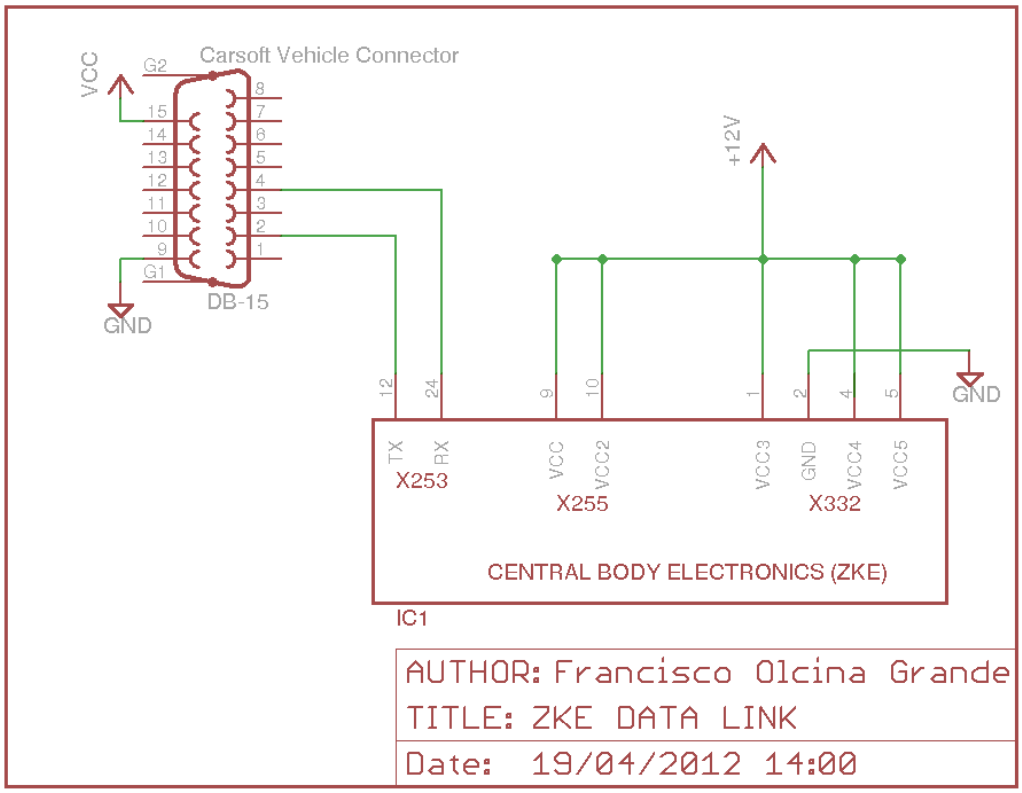


Figura 4.4: Conexiones mínimas para realizar una comunicación con la centralita ZKE

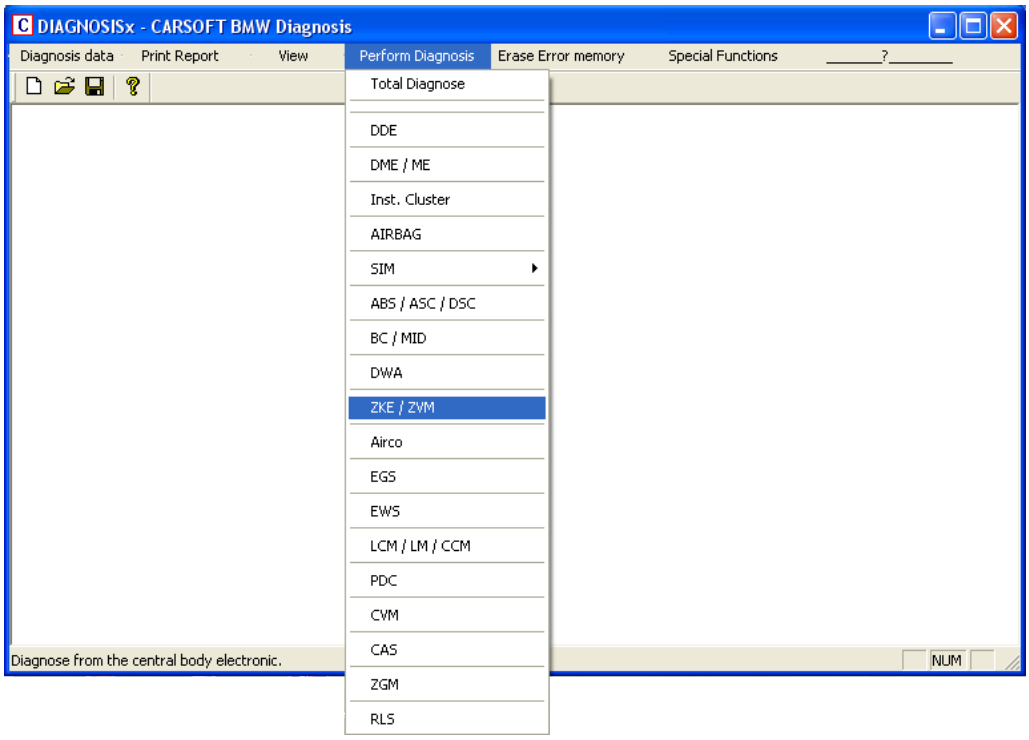


Figura 4.5: Selección de centralita para realizar una diagnosis en Carsoft.

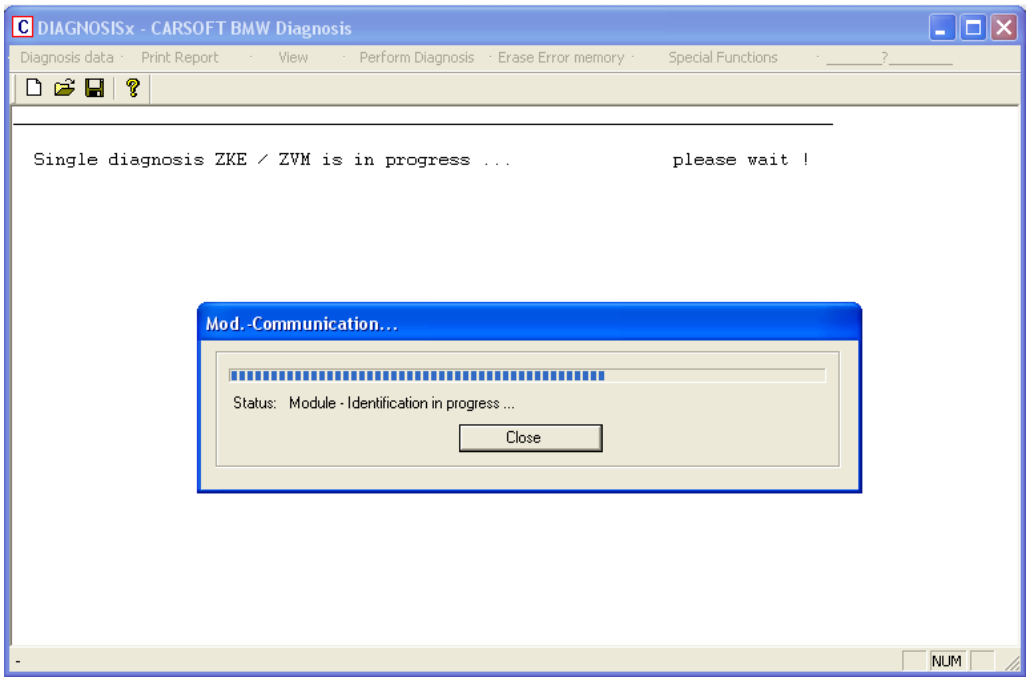


Figura 4.6: Identificación de la centralita mediante Carsoft.

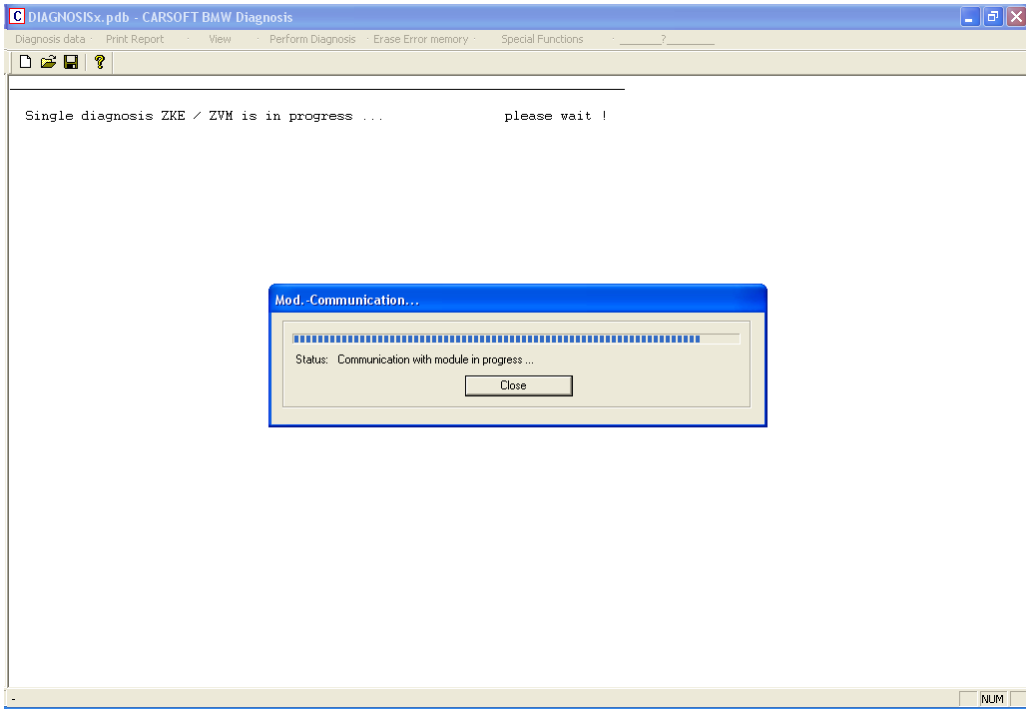


Figura 4.7: Comunicación con la centralita mediante Carsoft.

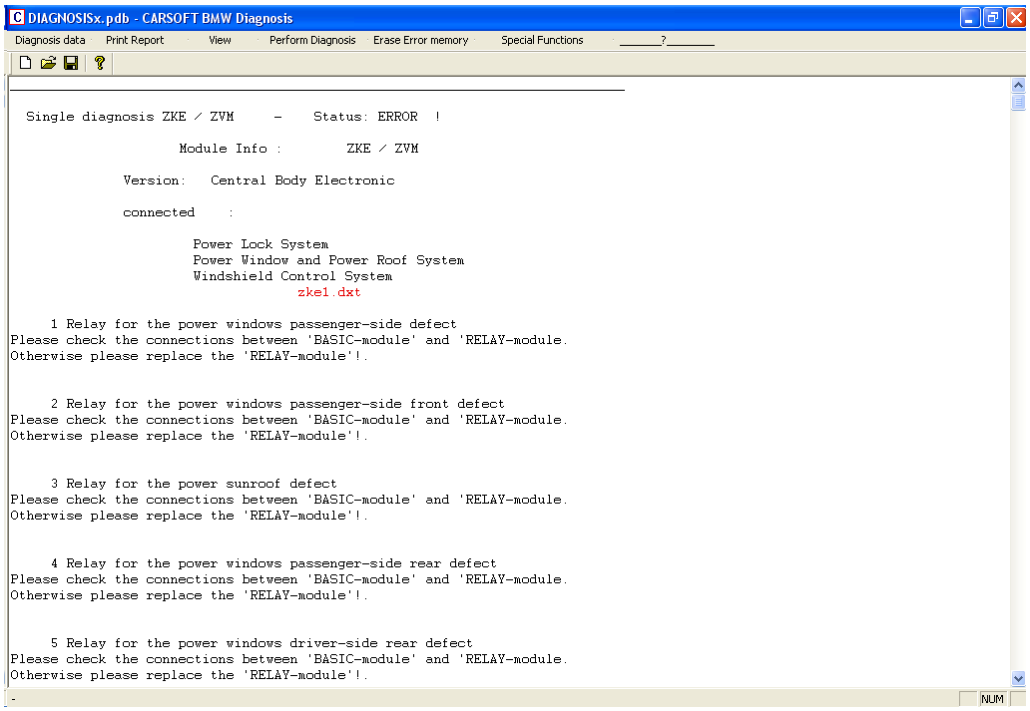


Figura 4.8: Resultado de la diagnosis con Carsoft de la centralita ZKE.

El resultado final de la diagnosis mostrado en la figura 4.8 es el siguiente:

```
Single diagnosis ZKE / ZVM     -    Status: ERROR  !

   Module Info :        ZKE / ZVM

    Version:   Central Body Electronic

    connected    :

     Power Lock System
     Power Window and Power Roof System
                    Windshield Control System
          zke1.dxt

1 Relay for the power windows passenger-side defect
Please check the connections between 'BASIC-module' and 'RELAY-module.
Otherwise please replace the 'RELAY-module'!


2 Relay for the power windows passenger-side front defect
Please check the connections between 'BASIC-module' and 'RELAY-module.
Otherwise please replace the 'RELAY-module'!


3 Relay for the power sunroof defect
Please check the connections between 'BASIC-module' and 'RELAY-module.
Otherwise please replace the 'RELAY-module'!


4 Relay for the power windows passenger-side rear defect
Please check the connections between 'BASIC-module' and 'RELAY-module.
Otherwise please replace the 'RELAY-module'!

5 Relay for the power windows driver-side rear defect
Please check the connections between 'BASIC-module' and 'RELAY-module.
Otherwise please replace the 'RELAY-module'!




  - Single diagnosis completed !
```

Se puede confirmar que ha funcionado la conexión dado que la centralita ha respondido a la diagnosis comunicando que no hay conexión con los elementos que controla, algo obvio dado que sólo se han realizado las conexiones mínimas para que la centralita responda. Por tanto, se ha alcanzado el hito de reproducir la comunicación con la centralita ZKE en una mesa de trabajo. En las siguiente sección se hace un análisis más en profundidad de la comunicación que establece Carsoft.

4.3. Análisis de la comunicación con Carsoft

El objetivo de esta sección es la analizar la comunicación que realiza el software y hardware de Carsoft y comprobar la viabilidad de reproducir dicha comunicación con otro software y/o hardware. Para ello se han dado los siguientes pasos:

- Analizar los datos enviados por el puerto serie.
- Enviar los datos analizados anteriormente con otro software.
- Comprobar los resultados obtenidos en el punto anterior para analizar el comportamiento del interfaz hardware.

4.3.1. Captura

Se ha utilizado el software Serial Port Monitor 4.0 de Eltima Software para analizar el comportamiento del puerto serie cuando se realiza una operación con Carsoft sobre la centralita. Del conjunto de operaciones que Carsoft puede realizar en el apartado denominado *Component Activation ZKE* (ver 2.11 en página 18), se ha seleccionado la operación de bajar la ventanilla del conductor y se han capturado las tramas de bytes enviadas al interfaz hardware para realizar la operación. En la figura 4.9 aparece el programa Serial Port Monitor analizando el puerto serie mientras se ejecuta Carsoft.

Las tramas de bytes capturadas se dividen en 4 bloques distintos en los que se abre y cierra el puerto COM. Estos bloques son los siguientes:

Primer bloque

Se abre el puerto COM y se configura con los siguientes parámetros:

- Baudios por segundo: 19.200.
- Bits de stop: 0.
- Bit de paridad: ninguno.
- Bits de datos: 8.

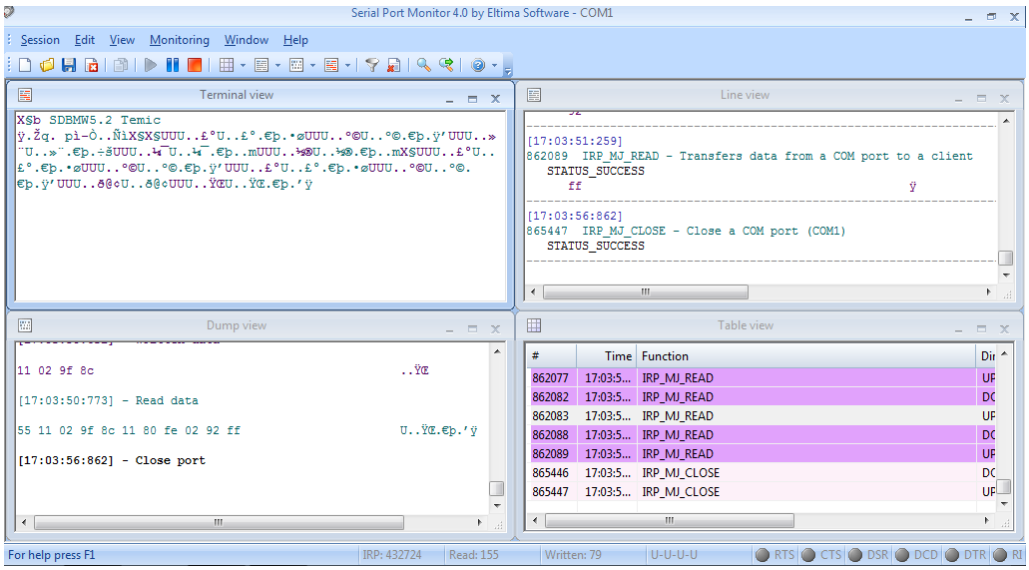


Figura 4.9: Monitorización del puerto serie mientras se realiza una operación con Carsoft.

Después se envían y reciben los siguientes bytes:

```
Envía: 58
Recibe: a7
```

Finalmente se cierra el puerto COM.

Segundo bloque

Se abre el puerto COM y se configura con los mismos parámetros que en el bloque anterior, después se envían y reciben los siguientes bytes:

```
Envía: 62
Recibe: 9D 53 44 42 4D 57 35 2E 32 20 54 65 6D 69 63 20
Envía: FF
Recibe: 00
Envía: 8E
Recibe: 71
Envía: 13 8F
Recibe: 70
Envía: EC 2D
Recibe: D2 13
Envía: 2E
Recibe: D1 EC
Envía: 58
Recibe: A7
```

Finalmente se cierra el puerto COM.

Tercer bloque

Se abre el puerto COM y se configura con los parámetros anteriores, después se envían y reciben los siguientes bytes:

```
Envía: 58
Recibe: A7
```

Sin cerrar el puerto se reconfigura con los siguientes parámetros:

- Baudios por segundo: 9.600.
- Bits de stop: 2.
- Bit de paridad: par.
- Bits de datos: 8.

Después se envían y reciben los siguientes bytes:

```
Envía: 55 55
Recibe: 55
Envía: 11 02 A3 B0
Recibe: 55 11 02 A3 B0 11 80 FE 02 95 F8
Envía: 55 55
Recibe: 55
Envía: 11 02 BA A9
Recibe: 55 11 02 BA A9 11 80 FE 02 FF 92
Envía: 55 55
Recibe: 55
Envía: 11 02 BB A8
Recibe: 55 11 02 BB A8 11 80 FE 02 F7 9A
Envía: 55 55
Recibe: 55
Envía: 11 02 BC AF
Recibe: 55 11 02 BC AF 11 80 FE 02 00 6D
Envía: 55 55
Recibe: 55
Envía: 11 02 BD AE
Recibe: 55 11 02 BD AE 11 80 FE 02 00 6D
```

Finalmente se cierra el puerto COM.

Cuarto bloque

Se abre de nuevo el puerto COM y se configura con los siguientes parámetros:

- Baudios por segundo: 19.200.

- Bits de stop: 0.
- Bit de paridad: ninguno.
- Bits de datos: 8.

Después se envían y reciben los siguientes bytes:

```
Envía: 58
Recibe: A7
```

Sin cerrar el puerto se reconfigura con los siguientes parámetros:

- Baudios por segundo: 9.600.
- Bits de stop: 2.
- Bit de paridad: par.
- Bits de datos: 8.

Después se envían y reciben los siguientes bytes:

```
Envía: 55 55
Recibe: 55
Envía: 11 02 A3 B0
Recibe: 55 11 02 A3 B0 11 80 FE 02 95 F8
Envía: 55 55
Recibe: 55
Envía: 11 02 BA A9
Recibe: 55 11 02 BA A9 11 80 FE 02 FF 92
Envía: 55 55
Recibe: 55
Envía: 11 02 A3 b0
Recibe: 55 11 02 A3 B0 11 80 FE 02 95 F8
Envía: 55 55
Recibe: 55
Envía: 11 02 BA A9
Recibe: 55 11 02 BA A9 11 80 FE 02 FF 92
Envía: 55 55
Recibe: 55
Envía: 11 03 F0 40 A2
Recibe: 55 11 03 F0 40 A2
Envía: 55 55
Recibe: 55
Envía: 11 02 9F 8C
Recibe: 55 11 02 9F 8C 11 80 FE 02 92 FF
```

Finalmente se cierra el puerto COM y termina la captura.

Analizando las tramas de los cuatro bloques se pueden ver dos configuraciones distintas de conexión en el puerto serie. Hay que tener en cuenta que Carsoft debe

detectar el tipo de coche con el que se quiere realizar la conexión porque se trata de un software de diagnosis para múltiples modelos de vehículos BMW con distintas tecnologías de diagnosis. Esto podría explicar las dos configuraciones distintas con las que se comunica con el interfaz hardware.

Otro dato a tener en cuenta es que el tercer bloque se ha capturado en el momento que Carsoft indicaba que estaba comunicándose con la centralita (ya se había detectado entonces), para después aparecer el menú de opciones que pueden realizarse sobre la centralita ZKE. El cuarto bloque se ha capturado justo después de dar a la opción de bajar la ventanilla del conductor, es presumible indicar que el tercer bloque se corresponde con la primera comunicación funcional con la centralita, y el cuarto bloque se corresponde con la operación de bajar la ventanilla exclusivamente.

4.3.2. Reproducción

La reproducción se ha hecho con el mismo software de análisis (Serial Port Monitor 4.0), de dos formas distintas:

1. Se han realizado los mismos inicios y cierres de conexión que en la captura y se han enviado los mismos datos, comenzando por el primer bloque, llegando un punto en el que el interfaz no respondía igual que en la captura. Dado que se desconoce qué está realizando el interfaz hardware al no responder igual, se tiene que descartar ésta vía.

2. Se realiza sólo la reproducción a partir del tercer bloque, suponiendo que las primeras conexiones son sólo para verificar el modelo de coche y de centralita donde se está realizando la comunicación, y que el tercer y cuarto bloque inicializan la propia centralita y realizan la operación sobre la ventanilla. Al reproducir el envío, tampoco se obtiene la misma respuesta del interfaz hardware, teniendo que descartar como útil esta reproducción.

De los intentos de reproducción anteriores se puede deducir que no se han reproducido exactamente las conexiones que realiza Carsoft a pesar de respetar la misma configuración del puerto serie, los mismos inicios y cierres de conexión y los mismos datos enviados. Hay otros parámetros que no se han reproducido, como los límites de tiempo entre inicios y cierres de conexión u otras señales enviadas por las líneas RTS o DTR.

Estos intentos fallidos de reproducir la conexión que realiza Carsoft con su propio interfaz hardware muestran la dificultad de discernir las comunicaciones que van dirigidas sólo al interfaz (para inicializarlo o como método de protección ante copias), las que van enfocadas a detectar la centralita a usar, y las que están dirigidas directamente a la centralita.

El resultado del análisis de las comunicaciones que establece Carsoft es indeterminado, por lo tanto se archivará y en la siguiente sección se avanzará en otra dirección: utilizando el software de fábrica de BMW y una reproducción compatible

de su interfaz hardware.

4.4. Utilización de EDIABAS ToolSet 32

4.4.1. Interfaz ADS

Como se comentó en el Capítulo del Estado de la Cuestión, el vehículo BMW E34 del proyecto no posee un conector OBD II como los vehículos actuales, sino el conector OBD I de 20 pines especial de BMW. Por otro lado, el software de BMW está adaptado para utilizar dos tipos de interfaz hardware en la conexión del equipo de diagnosis al vehículo: los interfaces OBD (dirigidos a los vehículos equipados con OBD II), y los interfaces ADS, dirigidos a los vehículos más antiguos de BMW con capacidad de diagnosis.

El tipo de interfaz que necesita el vehículo de pruebas es ADS o uno compatible. En la documentación de BMW, se detallan los esquemas para construir un interfaz de este tipo [1], pero debido a la complejidad de su construcción y que sólo se necesita compatibilidad con las centralitas del vehículo de pruebas, se construirá un interfaz similar pero con funcionalidad reducida adaptado a las necesidades del presente proyecto.

Análisis del funcionamiento del interfaz

El interfaz ADS original esta diseñado para adaptarse a todos los vehículos anteriores al OBD-II, por un lado se conecta al puerto serie del PC donde resida el software de diagnosis, y por otro lado se conecta al vehículo en su conector de 20 pines (OBD I). El puerto serie del PC ha de ser COM1 por restricciones del programa EDIABAS, el software que sirve de intérprete entre el resto de programas de BMW y el vehículo.

El interfaz ADS utiliza del puerto serie del PC las líneas RX, TX, DSR, DTR, RTS y RI. El cometido de cada línea es el siguiente:

- **Línea RX**: sirve para recibir los bytes de respuesta de las centralitas.
- **Línea TX**: sirve para enviar bytes a las centralitas.
- **Línea DSR**: sirve para comprobar si está dado el contacto en el vehículo.
- **Línea DTR**: sirve para conmutar el envío de bytes a las centralitas. Con un "1" lógico (-12V) envía los bytes a la línea RX de las centralitas más antiguas, o a la L de las centralitas más modernas. Con un "0" lógico (+12V) envía los bytes a la línea TX de las centralitas más antiguas, o a la línea K de las centralitas más modernas.
- **Línea RTS**: sirve para generar pulsos en la línea RX o L para inicializar las centralitas.

- **Línea RI**: sirve para comprobar si la batería está dando los 12V de tensión.

A su vez, el interfaz ADS utiliza del conector OBD I del vehículo las líneas +12V, Batería, GND, *Reset*, RX y TX (ver 2.1 en página 15).

El interfaz ADS trabaja para dos tipos de buses de datos:

- **Líneas RX y TX**: es el tipo más antiguo de línea. Son unidireccionales (*simplex*), las centralitas reciben los datos del PC a través de RX y los envían al PC a través de TX.
- **Líneas L y K**: son líneas más modernas que cumplen el estándar ISO-9141. La línea L es unidireccional, a través de ella las centralitas reciben datos o bien pulsos de inicialización; La línea K es bidireccional, a través de ella las centralitas pueden enviar y recibir datos.

Diseño de un interfaz ADS reducido

En el vehículo de pruebas todas las centralitas están conectadas a las líneas de datos RX y TX, incluida la centralita ZKE (según los esquemas de Bentley [18]). Dado que son líneas unidireccionales, no es necesario conmutar el envío desde el PC a una de las líneas como sucede en L y K, donde ambas pueden recibir el dato según la centralita y/o operación a realizar. El envío del PC siempre será a la línea RX de la centralita, y la respuesta de ésta última será a través de su línea TX. Esto permite reducir el diseño del interfaz ADS al omitir la conmutación, y por tanto la utilización de la línea DTR del puerto serie del PC, que es la encargada de ésta tarea.

Con el objetivo de construir un interfaz ADS más sencillo, y tomando como referencia otros diseños más actuales ([6] y [13]), se ha diseñado el interfaz compatible ADS de la figura 4.10 que permite comunicar el puerto serie de un PC con la centralita ZKE de la mesa de trabajo.

4.4.2. Herramienta EDIABAS ToolSet 32

La herramienta EDIABAS ToolSet 32 de BMW permite ejecutar cualquier operación del catálogo de operaciones que integra cada centralita. En primer lugar hay que seleccionar la centralita con la que se desea trabajar, en segundo lugar escoger la operación a realizar y en último lugar ejecutar dicha operación una sola vez o en bucle.

En la figura 4.11 aparecen las operaciones posibles sobre la centralita ZKE del vehículo de pruebas (fichero ZKE1.prg).

La traducción del alemán de las operaciones de la centralita ZKE es la siguiente:

- **Información**: información para el usuario sobre el modelo de centralita. No

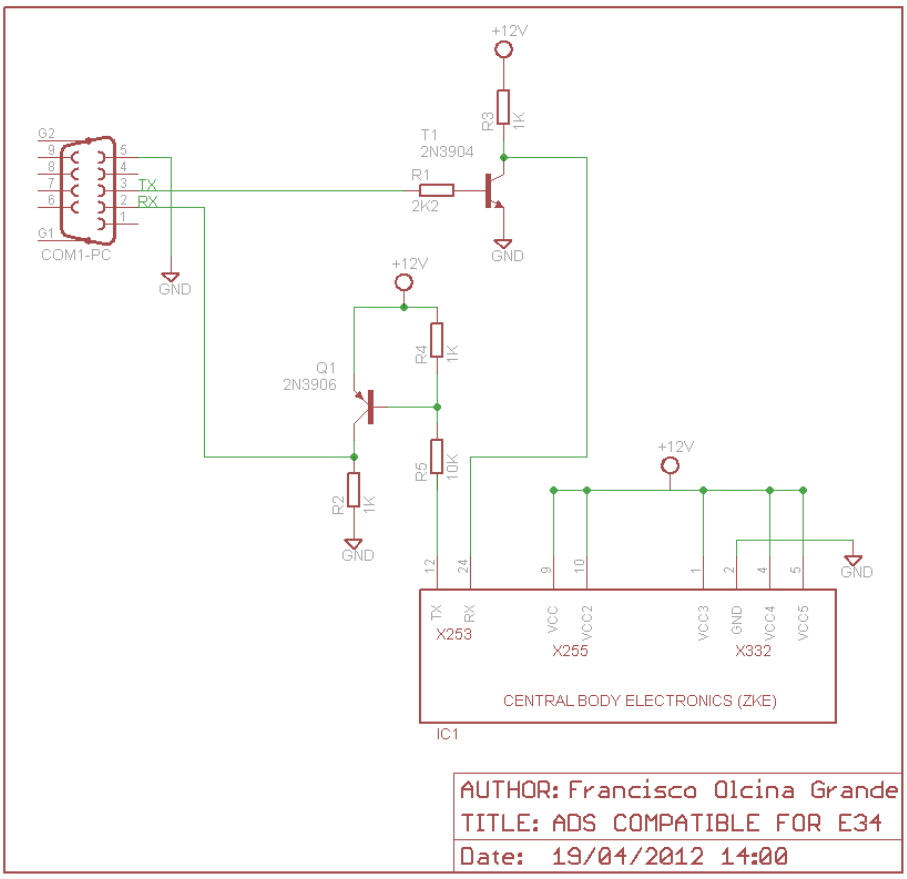


Figura 4.10: Interfaz compatible ADS

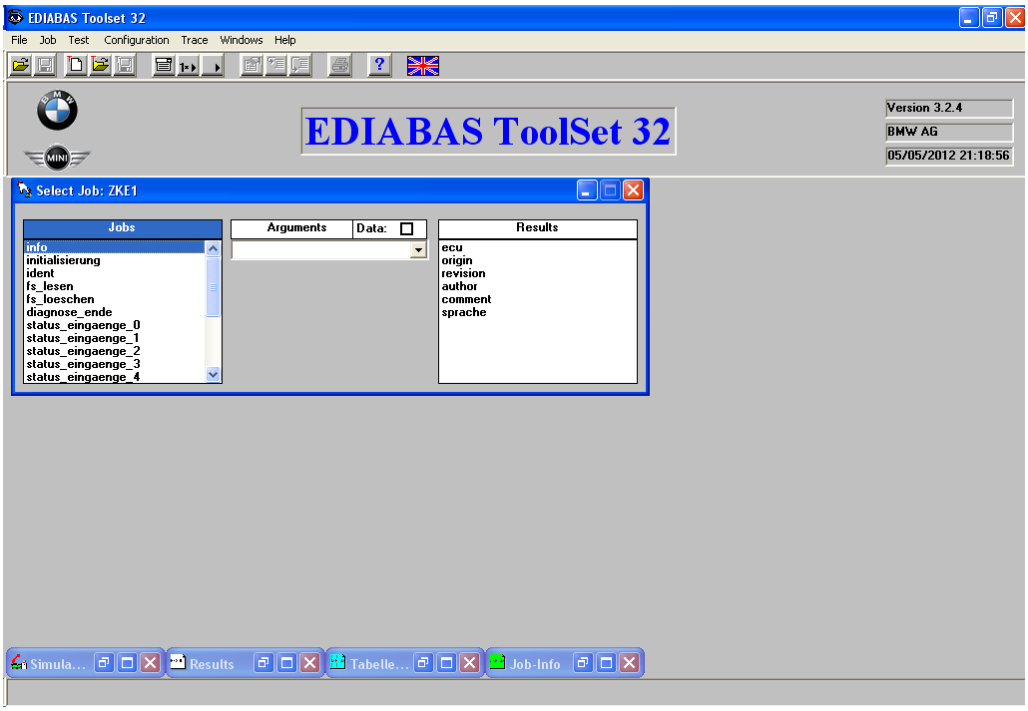


Figura 4.11: Operaciones de la centralita ZKE mostradas en ToolSet 32.

realiza ninguna conexión.

- **Inicialización**: llamada interna del programa cuando se selecciona la centralita.
- **Identidad**: obtiene de la centralita información variada (versión de software y hardware,...)
- **Lectura de la memoria de errores**: obtiene de la centralita los errores registrados en su memoria.
- **Borrado de la memoria de errores**: borra de la memoria de la centralita los errores registrados.
- **Finalizar diagnostico**: llamada interna del programa cuando se finaliza la operación con la centralita.
- **Estado de las entradas 0 a 8**: estado de las entradas digitales de la centralita (sensores de los motores de las ventanas, sensores de los cierres, etc.).
- **Estado de las salidas 0 a 2**: estado de las salidas digitales (si se ha pulsado algún botón de cierre, o de ventana, etc.)
- **Control de las entradas**: controla las entradas digitales permitiendo dar el valor deseado a las mismas.
- **Control de las salidas**: controla las salidas digitales, permitiendo dar el valor deseado a las mismas.

Para proceder al análisis de ToolSet 32, se ha conectado el interfaz ADS al PC y a la centralita ZKE de la mesa de trabajo, después se ha configurado la herramienta ToolSet 32 para que realice una de las operaciones del catálogo de la centralita y se ha observado la respuesta. La operación seleccionada ha sido *ident*, dando como resultado en la pantalla de la herramienta:

```
apiJob("ZKE1","ident","","")

Satz : 0
  OBJECT                         = zke1
  SAETZE                         = 1
  JOBNAME                        = ident
  VARIANTE                       = ZKE1
  JOBSTATUS                      =
  UBATTCURRENT                   = 0
  UBATTHISTORY                   = 1
  IGNITIONCURRENT                = 0
  IGNITIONHISTORY                = 1
Satz : 1
  JOB_STATUS                     = OKAY
  ID_IDENT                       = 1                 00 01  ..
```

```
ID_HW_NR                    = 0                 00 00   ..
ID_SW_NR                    = 0                 00 00   ..
ID_PROGSPEICHER             = 0                 00 00   ..
ID_AUSSTATTUNG              = 73                00 49   .I
```

La figura 4.12 muestra una captura de pantalla de la operación realizada.

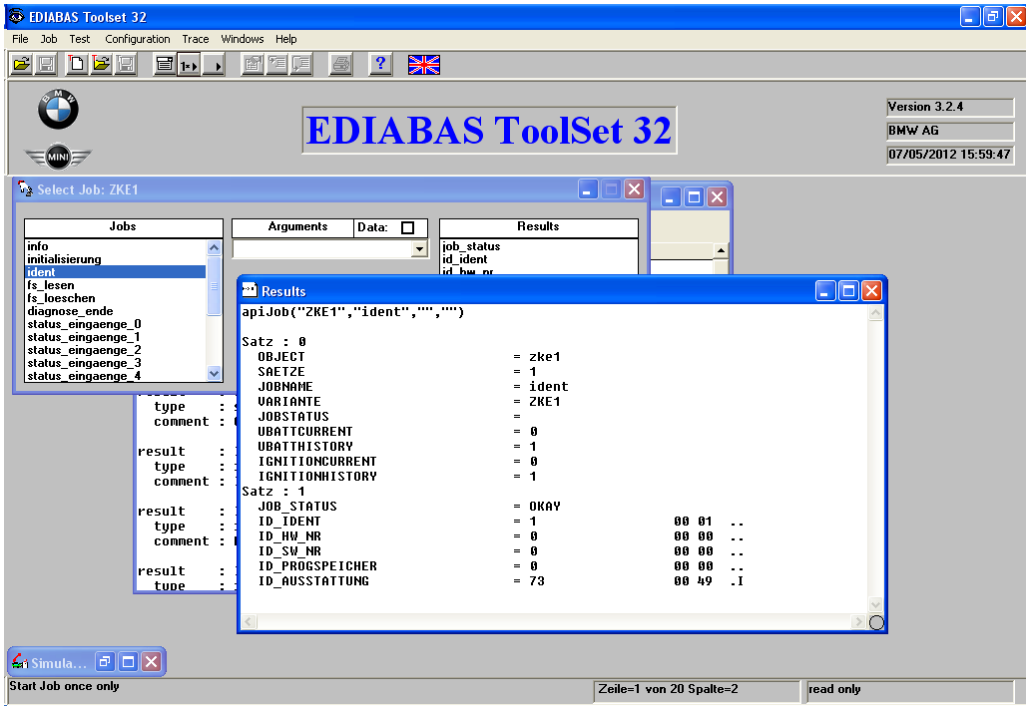


Figura 4.12: Operación *ident* realizada con ToolSet 32 sobre la centralita ZKE.

4.4.3. Captura y reproducción, primera iteración

Para capturar la comunicación entre el PC y la centralita, se ha hecho uso del software Serial Port Monitor 4.0 de Eltima Software de manera similar a cómo se procedió con el software Carsoft (ver apartado 4.3.1 en página 67). Se ha capturado el tráfico del puerto serie COM1 utilizado por la herramienta ToolSet 32 para el interfaz ADS obteniéndose la siguiente información:

- **Parámetros de la conexión**: 9.600 baudios/segundo, 8 bits de datos, 2 bits de stop, paridad par.
- **Bytes enviados**: ninguno.
- **Bytes recibidos**:

  ```
  11 80 FE 02 95 F8
  ```

Lo más destacable de la captura mediante Serial Port Monitor es que no se llegan a ver los bytes enviados, como muestra la figura 4.13, únicamente se capturan los bytes recibidos por el puerto. Para descartar la posibilidad de que Serial Port

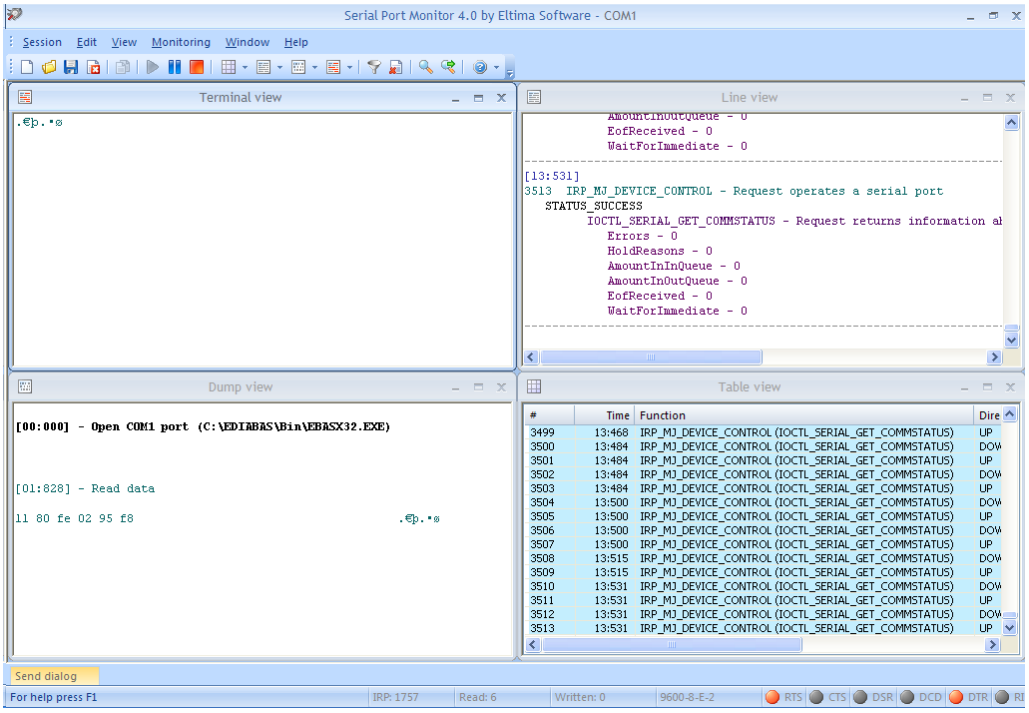


Figura 4.13: Captura de pantalla de Serial Port Monitor monitorizando COM1 mientras se utiliza ToolSet 32.

Monitor hay funcionado de forma incorrecta y por ello no capturase los bytes que ToolSet 32 emitía por el puerto serie, se ha repetido el procedimiento de captura con otras dos herramientas distintas: Advanced Serial Port Monitor de AGGsoftware y Free Serial Port Monitor de HHD Software. Con estas dos herramientas se ha alcanzado el mismo resultado, sólo se han capturado los bytes recibidos por el puerto serie.

Como resultado de las capturas, se concluye que ToolSet32 utiliza un sistema de envío de bytes que no puede capturarse con los programas de monitorización del puerto serie, por tanto, es necesario utilizar otro método para conseguir esa información.

Habilitando el modo de traza en ToolSet 32

Dado que el software ToolSet 32 permite habilitar una traza de los bytes enviados y recibidos, se ha habilitado esta opción y se ha realizado la misma operación, encontrando en el fichero de traza la siguiente información:

```
_ADS       INPUT: 55 55 11 02 A3
_ADS       ERROR: NO ERROR
_ADS       OUTPUT: 11 80 FE 02 95 F8
```

En la traza aparecen tanto los bytes enviados como los bytes recibidos al realizar la operación *ident* con ToolSet 32. Cabe destacar que los bytes recibidos coinciden

con los capturados por Serial Port Monitor y que al tener conocimiento de los bytes enviados, se puede reproducir la comunicación de forma manual.

Reproducción de la comunicación utilizando los bytes de la traza

El siguiente paso ha sido reproducir el envío de los bytes que refleja la traza de ToolSet 32 mediante el terminal que integra Serial Port Monitor. Para ello se ha habilitado el puerto COM1 con los siguientes parámetros de conexión: 9.600 baudios/segundo, 8 bits de datos, 2 bits de stop, paridad par.

Se ha habilitado también el modo de monitorización del puerto COM1 y se ha enviado la misma trama de bytes que aparecen en la traza anterior:

```
55 55 11 02 A3
```

Después del envío, no se ha obtenido ninguna respuesta.

4.4.4. Captura y reproducción, segunda iteración

Captura mediante un cable *sniffer*

Para asegurarse que la traza generada por ToolSet 32 no esté incompleta y como método más riguroso de obtener la información que fluye por el puerto COM1 del PC, se ha hecho uso de un cable serie a modo de *sniffer* conectado a otro puerto COM.

Este cable que actúa de *sniffer* recoge en su línea RX lo que se envía y recibe del puerto COM1 del ordenador, es decir, conecta las líneas RX y TX del puerto COM1 a la línea RX del cable *sniffer* de tal forma que no afecte a la comunicación del puerto COM1 (mediante diodos). El esquema de la conexión del cable *sniffer* entre el puerto COM1 y el interfaz ADS se muestra en la figura 4.14.

Dado que los PCs suelen tener sólo un puerto COM físico y en este caso esta siendo utilizado por el interfaz ADS, el cable *sniffer* debe conectarse al PC a través de un conversor Serie-USB.

El puerto serie virtual que crea el PC al utilizar el conversor Serie-USB es COM3. Para capturar el tráfico se ha utilizado de nuevo el terminal de Serial Port Monitor pero esta vez abriendo y monitorizando el puerto COM3. Los parámetros de apertura del puerto siguen siendo los mismos: 9.600 baudios/segundo, 8 bits de datos, 2 bits de stop, paridad par.

Con el puerto COM3 abierto y monitorizado, se ha vuelto a ejecutar la función *ident* en ToolSet32, capturándose las siguientes tramas de bytes:

```
55 55 11 02 A3 B0 11 80 FE 02 95 F8
```

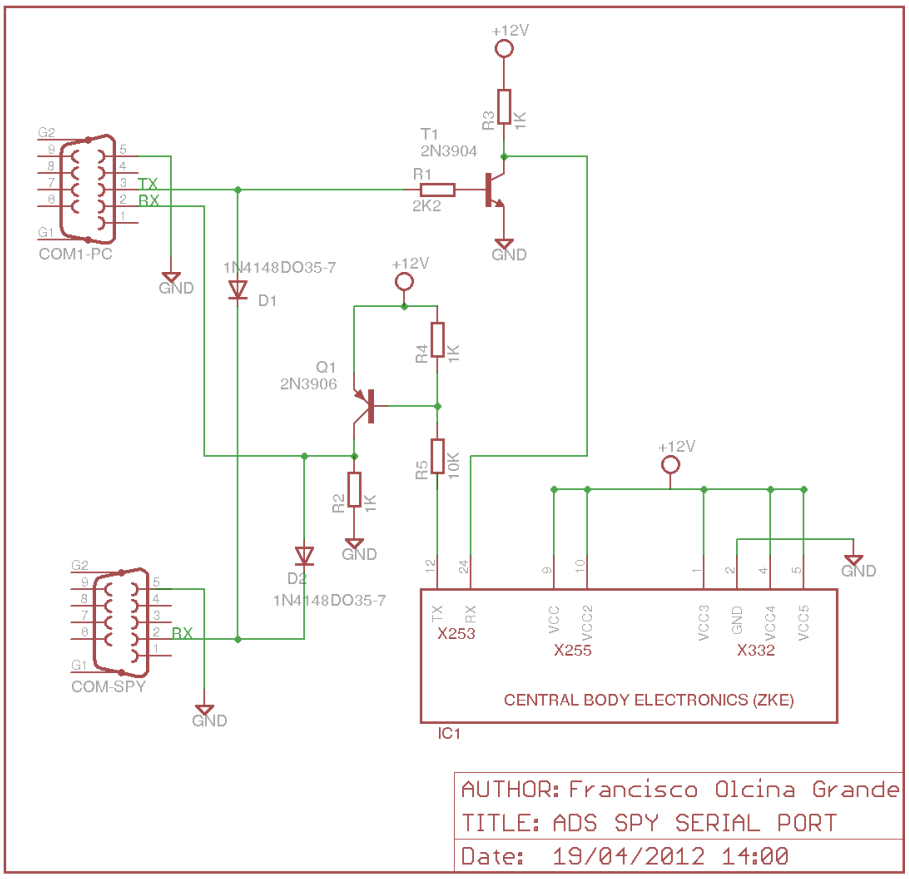


Figura 4.14: Captura del tráfico del puerto serie en el interfaz ADS

Esta trama se corresponde con los bytes del envío de la petición a través del puerto COM1 y la respuesta dada por la centralita y recibida por el mismo puerto (a través del interfaz ADS). Como muestra la figura 4.15, no hay distinción entre la petición y el envío en la captura porque se está capturando la información por una sola línea del puerto COM3.

En capturas realizadas en los pasos anteriores, se supo que los bytes de respuesta de la centralita son:

```
11 80 FE 02 95 F8
```

Por tanto, el resto de bytes de ésta última captura deben de pertenecer a la petición realizada. El cuadro 4.2 muestra la comparativa entre los bytes de la captura obtenida con el cable *sniffer*, con los bytes de la traza de ToolSet 32. En la comparativa se aprecia un byte de diferencia entre las peticiones: la captura con el cable *sniffer* ha registrado un byte adicional con valor "B0" que la traza de ToolSet 32 ha omitido.

Buscando una explicación a ésta omisión en la traza de ToolSet 32, se ha planteado la hipótesis de que el byte omitido puede pertenecer a algún tipo de código de corrección de errores. Para confirmar esta hipótesis, se ha realizado la siguiente comprobación: utilizando los bytes de cada trama salvo el último byte, y

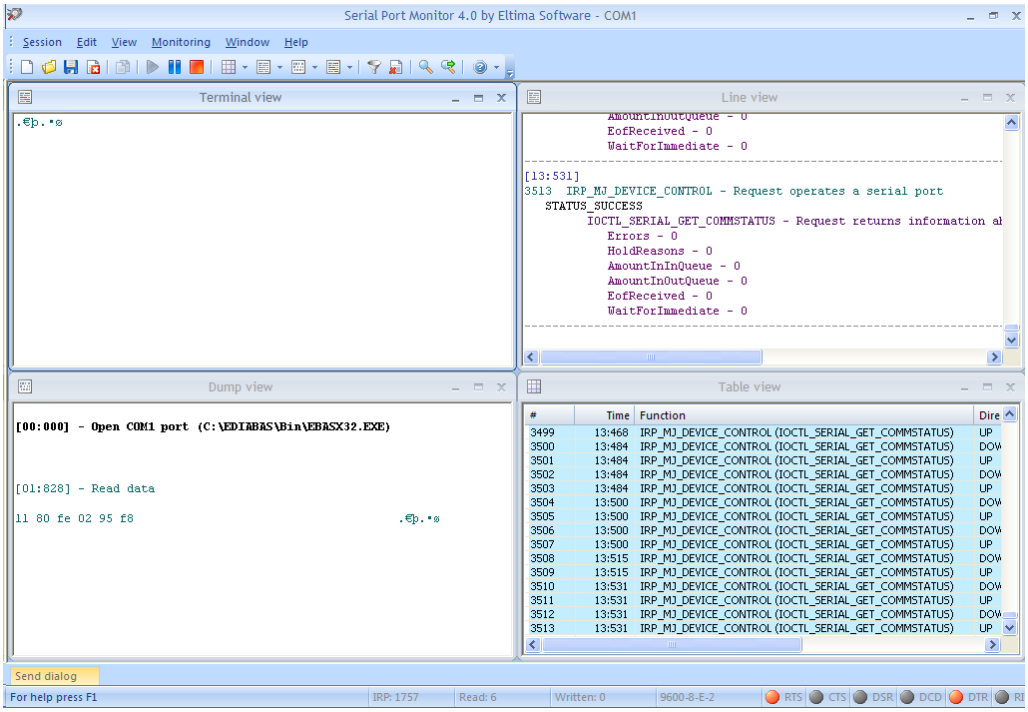


Figura 4.15: Captura de pantalla de Serial Port Monitor monitorizando con un cable *sniffer* la comunicación de ToolSet 32.

	Traza de ToolSet 32	**Captura con el cable *sniffer***
Petición	55 55 11 02 A3	55 55 11 02 A3 B0
Respuesta	11 80 FE 02 95 F8	11 80 FE 02 95 F8

Cuadro 4.2: Comparativa entre la traza de ToolSet32 y la captura con el cable *sniffer*.

realizando las operaciones de detección de errores más simples, se comprobó que el último byte pertenecía a uno de éstos códigos de detección, en concreto se trataba de un chequeo de paridad horizontal o longitudinal, conocido como LRC (*Longitudinal Redundancy Check*) o HRC (*Horizontal Redundancy Check*) [1].

Reproducción de los bytes capturados

De nuevo se ha intentado reproducir el envío de las tramas de bytes mediante Serial Port Monitor incluyendo en esta ocasión el byte de CRC. En la figura 4.16 se puede observar la herramienta Serial Port Monitor configurando el puerto COM1 con los mismos parámetros que en los casos anteriores, (9.600 baudios/segundo, 8 bits de datos, 2 bits de stop, paridad par) y listo para enviar la trama de bytes:

```
55 55 11 02 A3 B0
```

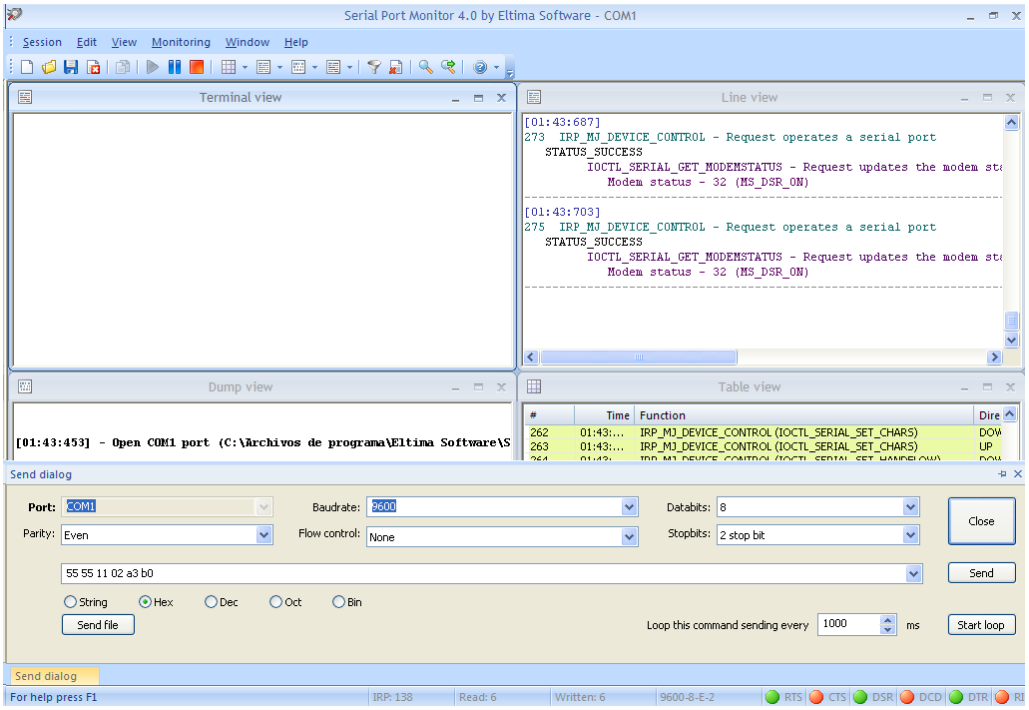


Figura 4.16: Captura de Serial Port Monitor listo para enviar una trama de bytes por el puerto COM1.

Como se puede observar en la figura 4.17, al enviar los bytes anteriores, la centralita respondió con la siguiente trama de bytes:

[1] El chequeo de paridad horizontal o longitudinal (HRC ó LRC) en vez de estar orientado al carácter lo está al mensaje, y consiste en que cada posición de bit de un mensaje tiene bit de paridad, así por ejemplo se toman todos los bits b0 de los caracteres que componen el mensaje y se calcula un bit de paridad par o impar, según el criterio definido, este bit de paridad es el bit b0 de un carácter adicional que se transmite al final del mensaje, y se procede luego sucesivamente con los demás bits incluyendo el de paridad. El carácter así construido se denomina BCC (*Block Check Character*), también se le denomina BCS (*Block Character Sequence*)[17].

```
11 80 FE 02 95 F8
```

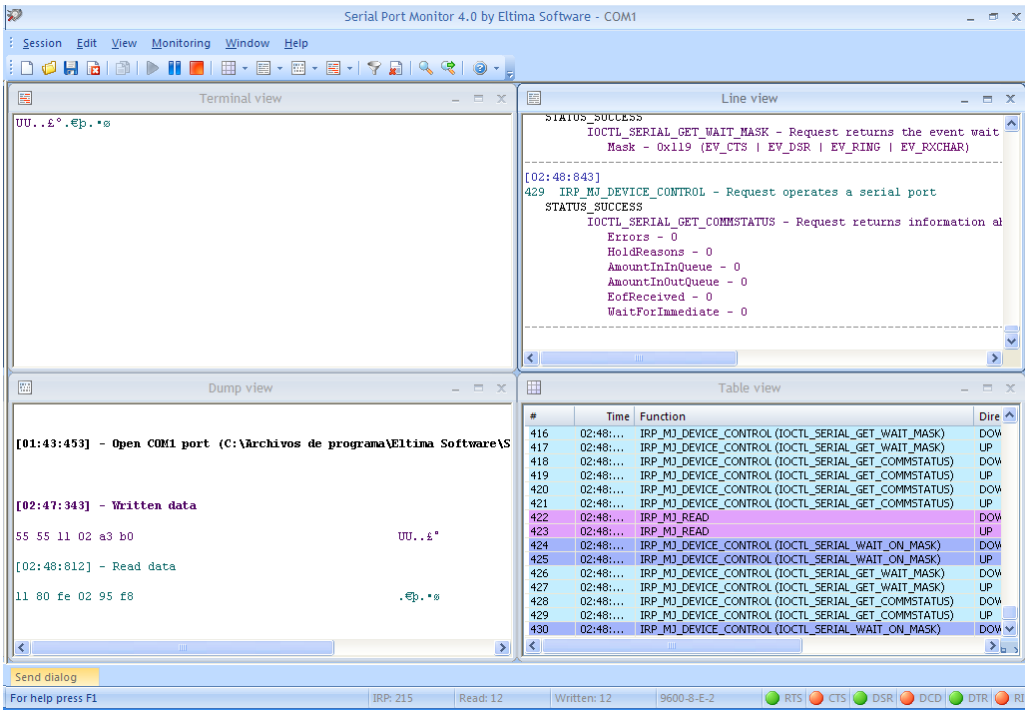


Figura 4.17: Captura de Serial Port Monitor mostrando la respuesta de la centralita ZKE.

que es la misma respuesta obtenida mediante ToolSet 32, por tanto la reproducción ha sido exitosa.

4.5. Creación de un diccionario

El objetivo de esta sección es el de crear un diccionario de tramas de bytes con las funciones de la centralita ZKE que controlan los elementos del vehículo (ventanas, techo, cierre y limpiaparabrisas). Se pueden obtener dichas tramas a través de los dos programas utilizados en las secciones anteriores, pero cada uno presenta distintas ventajas e inconvenientes:

1. Mediante EDIABAS ToolSet 32 es fácil realizar operaciones y revisar las tramas a través de su modo de depuración, o en su defecto, con el cable *sniffer*; no obstante para ejecutar las acciones buscadas sobre los elementos del vehículo sólo existe la función "Control de las salidas", que requiere de varios parámetros desconocidos.

2. Mediante Carsoft se pueden ejecutar directamente las funciones buscadas a través de la ventana "Component Activation ZKE" (ver figura 2.11 en página 18), pero en las capturas realizadas en la sección 4.3 (ver página 67) ejecutando una de las operaciones de la centralita, no se llegó a constatar si eran tramas enviadas al interfaz hardware de Carsoft e interpretadas posteriormente, o eran las tramas que directamente se enviarían al vehículo y por tanto las necesarias para el diccionario.

Para alcanzar el objetivo de ésta sección se han usado las dos herramientas para poder contrastar la información obtenida de cada una de ellas.

4.5.1. Tramas de Carsoft

Se han utilizado las tramas capturadas en Carsoft en la sección 4.3 (ver página 67) y se han analizado de nuevo teniendo en cuenta el conocimiento adquirido con ToolSet 32.

En concreto se ha reutilizado la captura del cuarto y último bloque de las capturas mencionadas, perteneciente teóricamente a la comunicación con la centralita ZKE para realizar la acción de bajar la ventanilla del conductor.

Captura realizada con Serial Port Monitor

Los parámetros de conexión iniciales en la captura del cuarto y último bloque fueron: 19.200 baudios por segundo, 8 bits de datos, 1 bit de stop, sin paridad. Se capturaron los siguientes bytes:

```
Envía: 58
Recibe: A7
```

Después, sin cerrar el puerto, se cambiaron los parámetros de la conexión por estos otros: 9.600 baudios/segundo, 8 bits de datos, 2 bits de stop, paridad par, que corresponden a los mismos parámetros que utiliza la centralita ZKE, induciendo a

pensar que los bytes anteriores sólo estaban dirigidos al interfaz hardware de Carsoft, no a la centralita. Los bytes que se capturaron con éstos últimos parámetros fueron:

```
Envía: 55 55
Recibe: 55
Envía: 11 02 A3 B0
Recibe: 55 11 02 A3 B0 11 80 FE 02 95 F8
Envía: 55 55
Recibe: 55
Envía: 11 02 BA A9
Recibe: 55 11 02 BA A9 11 80 FE 02 FF 92
Envía: 55 55
Recibe: 55
Envía: 11 02 A3 B0
Recibe: 55 11 02 A3 B0 11 80 FE 02 95 F8
Envía: 55 55
Recibe: 55
Envía: 11 02 BA A9
Recibe: 55 11 02 BA A9 11 80 FE 02 FF 92
Envía: 55 55
Recibe: 55
Envía: 11 03 F0 40 A2
Recibe: 55 11 03 F0 40 A2
Envía: 55 55
Recibe: 55
Envía: 11 02 9F 8C
Recibe: 55 11 02 9F 8C 11 80 FE 02 92 FF
```

Análisis de las tramas de Carsoft

En las tramas de la captura se ha observado lo siguiente:

- Se realiza una primera petición con el byte 58 y la respuesta A7. Éstas dos tramas se realizan con unos parámetros de conexión distintos a las tramas posteriores.
- Las siguientes peticiones que siempre comienzan con los bytes “55 55 11” y las respuestas incluyen un *echo* de la petición y otros bytes adicionales.

Comparando las observaciones de las tramas capturadas con Carsoft y con ToolSet 32 y también con la información sobre el protocolo DS-2 se ha llegado a las siguientes conclusiones:

- Tanto Carsoft como ToolSet 32 inician las peticiones a la centralita ZKE siempre con los mismos bytes: “55 55 11”. Éstos bytes corresponderían según DS-2 a la dirección de la centralita.

- Las respuestas de la centralita ZKE se inician siempre con los bytes "11 80 FE".
- La petición inicial de Carsoft con el byte "58" se dirige a su propio interfaz hardware, no a la centralita, por lo tanto puede descartarse con el interfaz compatible ADS que se utilizó para reproducir la comunicación mediante hyperterminal.
- La centralita ZKE no hace *echo* de la trama de bytes en la petición, así que los *echos* recibidos en la captura de Carsoft provienen del interfaz hardware del mismo.
- La petición cuyos bytes son "55 55 11 03 F0 40 A2" obtiene como respuesta sólo un *echo*, es decir, la centralita no responde nada según se afirmó en el punto anterior.
- En todas las peticiones a la centralita, el byte que sucede a la dirección de la centralita (byte posterior a "55 55 11") podría ser un byte de longitud, pero en las capturas realizadas solo se da esta situación contando los bytes que le suceden, no los bytes totales de la trama.

Tramas útiles de la captura de Carsoft

Extrayendo las tramas útiles (aquellas dirigidas a la centralita) de la captura realizada, se quedan las siguientes peticiones:

```
55 55 11 02 A3 B0
55 55 11 02 BA A9
55 55 11 02 A3 B0
55 55 11 02 BA A9
55 55 11 03 F0 40 A2
55 55 11 02 9F 8C
```

El siguiente paso es contrastar dichas tramas con las que genera ToolSet 32 para las distintas operaciones.

4.5.2. Tramas de ToolSet 32

Para recopilar las tramas que genera ToolSet32 y poder contrastarlas con las que genera Carsoft, se ha vuelto a utilizar el cable *sniffer*.

De las operaciones existentes, se ha capturado el tráfico sólo de aquellas que generan una comunicación con la centralita y que no necesitan parámetros para ejecutarse (se desconoce qué parámetros hay que enviar).

La tabla 4.3 refleja las tramas capturadas; aquellas que están en color gris aparecían en la captura de Carsoft anterior para la operación de subir la ventanilla.

Operación	**Tramas de las peticiones**
Identidad	55 55 11 02 A3 B0
Lectura de la memoria de errores	55 55 11 02 BC AF
	55 55 11 02 B8 AB
	55 55 11 02 BB A8
	55 55 11 02 B7 A4
	55 55 11 02 BA A9
	55 55 11 02 B6 A5
	55 55 11 02 BD AE
	55 55 11 02 B9 AA
Borrado de la memoria de errores	55 55 11 02 9F 8C
Estado de las entradas 0	55 55 11 02 A0 B3
Estado de las entradas 1	55 55 11 02 A1 B2
Estado de las entradas 2	55 55 11 02 A2 B1
Estado de las entradas 3	55 55 11 02 A3 B0
Estado de las entradas 4	55 55 11 02 A4 B7
Estado de las entradas 5	55 55 11 02 A5 B6
Estado de las entradas 6	55 55 11 02 A6 B5
Estado de las entradas 7	55 55 11 02 A7 B4
Estado de las entradas 8	55 55 11 02 A8 BB
Estado de las salidas 0	55 55 11 02 B0 A3
Estado de las salidas 1	55 55 11 02 B1 A2
Estado de las salidas 2	55 55 11 02 B2 A1

Cuadro 4.3: Tramas de bytes de distintas operaciones realizadas con ToolSet32.

4.5.3. Comparativa de las tramas de Carsoft y ToolSet 32

Partiendo de las tramas capturadas en Carsoft para la operación de bajar la ventanilla y de las tramas capturadas en ToolSet para otras funciones, se puede identificar en el primer caso la función de cada trama. En la tabla 4.4 se observa ésta relación.

Dado que todas las tramas han sido identificadas a través de ToolSet salvo la compuesta por los siguientes bytes:

```
55 55 11 03 F0 40 A2
```

Es presumible que sea la trama que realiza directamente la acción de bajar la ventanilla.

Bytes de la trama	**Función**
55 55 11 02 A3 B0	Identidad
55 55 11 02 BA A9	Lectura de errores
55 55 11 02 A3 B0	Identidad
55 55 11 02 BA A9	Lectura de errores
55 55 11 03 F0 40 A2	Posible función de bajar la ventanilla
55 55 11 02 9F 8C	Borrado de errores

Cuadro 4.4: Explicación de las tramas capturadas en Carsoft.

4.5.4. Metodología y diccionario final

Con la identificación de todas las tramas en la captura de Carsoft realizando la acción de bajar la ventanilla, se ha podido establecer una metodología para extraer las tramas que realizan el resto de operaciones de la centralita ZKE sobre los elementos del vehículo y que aparecen en el menú de Carsoft *Component Activation ZKE*.

La metodología ha sido la siguiente:

1. Abrir Carsoft utilizando el puerto COM1 y después la ventana *Component Activation ZKE*.
2. Comenzar a capturar el tráfico del puerto COM1 con Serial Port Monitor.
3. Realizar una de las operaciones de la ventana *Component Activation ZKE*.

4. De los bytes capturados con Serial Port Monitor, extraer las peticiones, descartar aquellas que pertenecen a las operaciones "identidad", "lectura de la memoria de errores" y "borrado de la memoria de errores" y anotar la trama que queda resultante, que será la que ejecute directamente la operación seleccionada en el punto 3.
5. Reiniciar la captura de Serial Port Monitor
6. Volver al punto 2 con otra de las operaciones restantes y si no quedan más realizar la tabla de correspondencia entre las tramas y las acciones que realizan.

La tabla 4.5 muestra el diccionario de tramas de bytes final.

Función	**Bytes de la trama**
Abrir ventana del conductor	55 55 11 03 F0 40 A2
Cerrar ventana del conductor	55 55 11 03 F0 A8 4A
Abrir ventana del pasajero	55 55 11 03 F0 04 E6
Cerrar ventana pasajero	55 55 11 03 F0 12 F0
Abrir ventana trasera del conductor	55 55 11 03 F0 20 C2
Cerrar ventana trasera del conductor	55 55 11 03 F0 C8 2A
Abrir ventana trasera del pasajero	55 55 11 03 F0 10 F2
Cerrar ventana trasera del pasajero	55 55 11 03 F0 06 E4
Abrir techo	55 55 11 03 F0 E0 02
Cerrar techo	55 55 11 03 F0 08 EA
Cerrar vehículo	55 55 11 03 F2 01 E1
Abrir vehículo	55 55 11 03 F2 02 E2
Cerrar vehículo con doble cierre	55 55 11 03 F2 05 E5
Activar el limpiaparabrisas	55 55 11 03 F1 21 C2

Cuadro 4.5: Diccionario de tramas de la centralita ZKE para realizar acciones sobre el vehículo.

Capítulo 5

Interfaz de conexión remota

5.1. Introducción

En este capítulo se detalla la construcción del interfaz de conexión remoto con el vehículo. El capítulo se divide en dos partes: una sección de diseño donde se detallan las distintas fases de diseño del circuito y una fase de implementación donde se elabora físicamente el interfaz de conexión remota.

5.2. Diseño

Ésta sección muestra las distintas fases de diseño del interfaz de conexión remota y se divide en los siguientes apartados:

- **Diagrama de despliegue**: donde se muestran los nodos físicos que componen el sistema completo y en particular el nodo del interfaz en detalle.
- **Módem Bluetooth**: donde se muestran las decisiones de diseño relativas al módem Bluetooth.
- **Microcontrolador I - Hardware**: donde se muestra el diseño relativo a la parte hardware del microcontrolador.
- **Microcontrolador I - Software**: donde se muestra el diseño relativo a la parte software del microcontrolador.
- **PCB (*Printed Circuit Board*)**: donde se muestra el diseño relativo al circuito impreso (PCB) utilizando la herramienta Eagle.

5.2.1. Diagrama de despliegue

La figura 5.1 muestra el diagrama de despliegue del sistema, donde aparecen los 3 nodos físicos principales: el dispositivo móvil (teléfono), el interfaz de conexión remota y por último el vehículo.

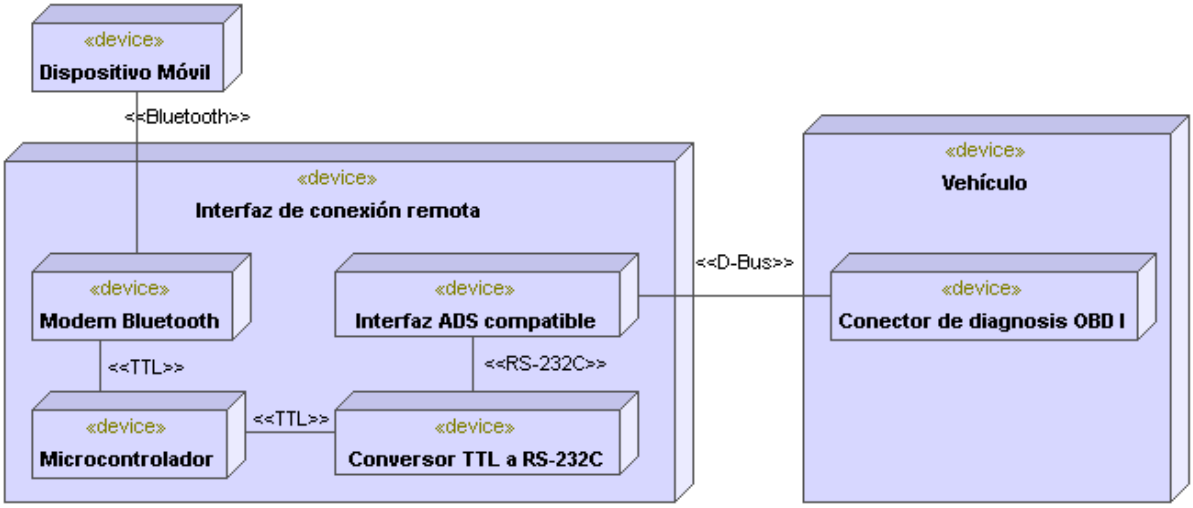


Figura 5.1: Diagrama de despliegue del sistema (interfaz de conexión remota detallado).

El nodo físico relativo al interfaz de conexión remota, tiene como objetivo recibir las peticiones que le lleguen por Bluetooth (desde el móvil) y enviarlas al conector de diagnosis del vehículo. El interfaz se compone de los siguientes nodos:

- **Módem Bluetooth**: es el encargado de comunicarse vía Bluetooth con el teléfono móvil que tendrá la aplicación de Android instalada. Los bytes que recibe el módem los enviará directamente al microcontrolador; a su vez, el microcontrolador también puede enviar bytes al módem y éste los transmitiría al teléfono móvil.
- **Microcontrolador**: actúa de puente entre el módem Bluetooth y el interfaz ADS, siendo éste último el encargado de adaptar los bytes del microcontrolador a los voltajes requeridos por el conector del vehículo.
- **Conversor TTL a RS-232C**: esta situado entre el microcontrolador y el interfaz ADS. Es el encargado de convertir las señales eléctricas de tipo TTL que gestiona la USART del microcontrolador, a señales eléctricas de tipo RS-232C que gestiona el interfaz ADS.
- **Interfaz ADS compatible**: es el encargado de adaptar las señales eléctricas RS-232C a las señales eléctricas que gestionan las líneas de datos de las centralitas del vehículo (líneas RX y TX).

Físicamente el interfaz es un circuito eléctrico sobre una placa de cobre que con-

tiene las partes anteriores y se conecta al vehículo por su toma de diagnosis.

Para la elaboración del diseño del circuito impreso se han tenido en cuenta las siguientes consideraciones:

- La placa final se ha de realizar manualmente: se ha utilizado una placa de cobre de una sola cara y componentes con encapsulados de tipo DIP, más fáciles de soldar que los de tipo SOIC o SMD.
- El voltaje se ha de obtener de la toma de diagnosis del vehículo, por tanto recibirá 12V de corriente continua. Es necesario un regulador de voltaje dentro del circuito para convertir los 12V a 5V, que son los que necesita el módem Bluetooth y el microcontrolador.

5.2.2. Módem Bluetooth

El módem Bluetooth que integra el circuito debe cumplir al menos con la clasificación "clase 2" que permite la comunicación en un radio de hasta 10m aproximadamente. Además es necesario que tenga un encapsulado de tipo DIP. El módem BlueSMiRF Silver de la compañía Sparkfun cumple con las necesidades anteriores y ofrece una buena relación calidad/precio ($39.95 en la página web del fabricante) [20]. En las figuras 5.2 y 5.3 se puede apreciar la cara superior e inferior del módem.

Las características del módem BlueSMiRF Silver son las siguientes:

- Clase 2, con un radio de 18 metros aproximadamente según el fabricante.
- Versión 2.0 con soporte EDR.
- Dimensiones: 45 x 16.6 x 3.9 mm.
- Conexión cifrada.
- Frecuencia entre 2.4 y 2.524 GHz.
- Voltaje de operación entre 3.3V y 6V.
- Comunicación serie en el rango de velocidad de 2.400 a 115.200 bps.
- Temperatura de trabajo entre -40 y +70C.
- Antena integrada.

Las conexiones que tiene el módem BlueSMiRF Silver enumeradas según la figura 5.3 de izquierda a derecha son las siguientes:

- **RTS**: línea de *Ready to Send*. Funciona como la línea homónima de un puerto serie de PC. No se utilizará en el circuito final.
- **RX**: línea de recepción de datos, a través de ella el módem recepcionará los bytes a enviar vía Bluetooth.

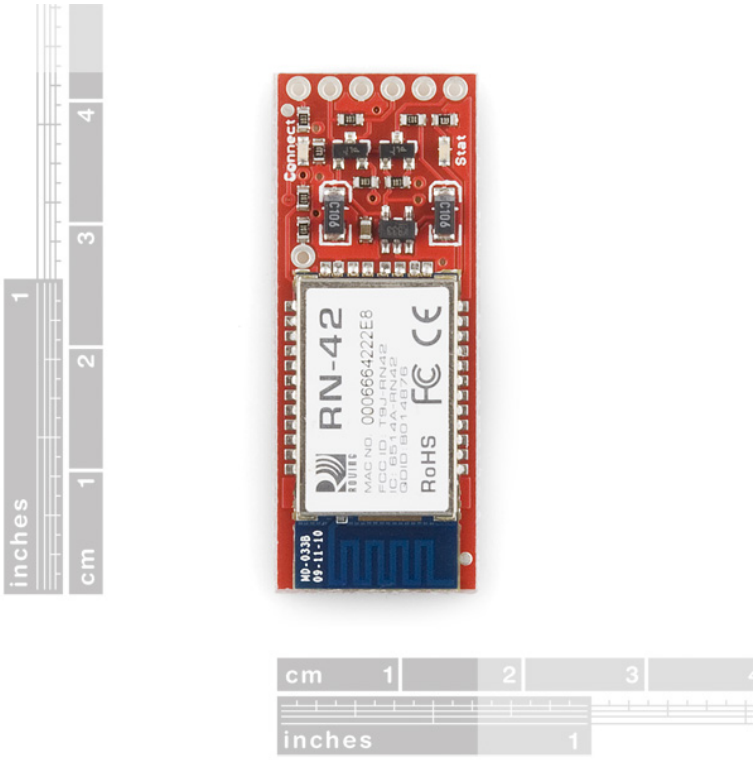


Figura 5.2: Cara superior del módem BlueSMiRF Silver.

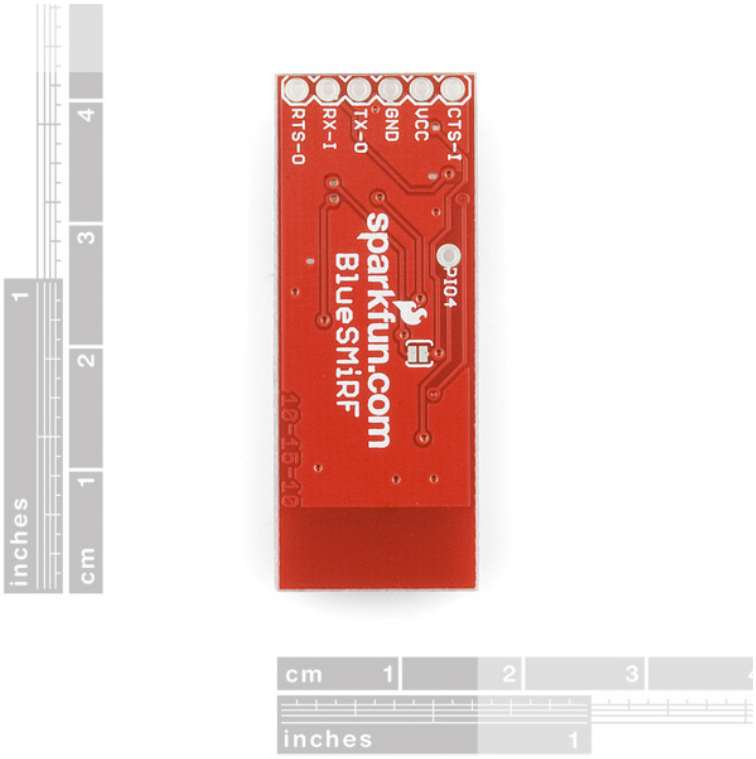


Figura 5.3: Cara inferior del módem BlueSMiRF Silver.

- **TX**: línea de envío de datos. A través de ella el módem envía los datos que ha recibido vía Bluetooth.
- **GND**: toma de tierra.
- **VCC**: toma de corriente positiva, o 3,3V o 5V.
- **CTS**: línea de *Clear to Send*. Funciona como la línea homónima de un puerto serie de PC. No se utilizará en el circuito final.

Las líneas RX y TX del módem se han de conectar a las líneas RX y TX respectivamente del microcontrolador. Ambos componentes funcionan bajo el estándar TTL y no requieren adaptadores de línea. Sin embargo, la toma de corriente (VCC) y tierra (GND) del módem requerirá del uso del regulador de 5V integrado en el circuito final.

5.2.3. Microcontrolador I - Hardware

La función del microcontrolador en el circuito final es la de hacer un diseño escalable que permita integrar cierta lógica dentro del propio circuito, independientemente del la lógica integrada en la aplicación remota. Inicialmente funciona como una tubería o *pipe* entre el módem Bluetooth y el interfaz ADS, pero reprogramándolo en futuras actualizaciones será capaz de realizar más funciones por si mismo.

Para actuar de *pipe* entre el módem y el interfaz ADS necesita de dos USARTS, ya sean físicas o virtuales (simuladas con entradas y salidas del propio microcontrolador): una de ellas se conectará con el módem Bluetooth y la otra con el interfaz ADS. Se ha buscado un microcontrolador que tenga al menos dos USARTS físicas para evitar la complejidad de tener que programar una de ellas sobre los pines de entrada y salida digital.

El microcontrolador escogido ha sido el AVR ATMega644A de la compañía ATMEL. Tiene las siguientes características según su *datasheet* [5]:

- Microcontrolador de 8 bits y 64 KBytes de memoria programable.
- Arquitectura RISC.
- Hasta 20 Mhz de frecuencia de reloj.
- Capacidad para ejecutar 1 MIPS por MHz.
- Dos *timers* o contadores de 8 bits y uno de 16 bits.
- Dos USARTS físicas.
- Seis modos de bajo consumo.
- Rango de voltajes operables entre 1.8V y 5.5V.
- Encapsulado de 40 pines de tipo DIP.

Elección de la frecuencia de reloj

Los componentes más importantes del AVR para el circuito final son las USARTs. Dado que funcionan como un *pipe*, recepcionando los datos del módem Bluetooth por una de ellas, y enviando dichos datos de forma inmediata por la otra, al interfaz ADS, obligatoriamente han de funcionar a la misma velocidad que requiera el protocolo DS-2, es decir, a 9.600 baudios.

Según el *datasheet* del AVR, dependiendo de la velocidad del reloj con la que trabaje el microcontrolador la generación interna de los baudios llevará un determinado porcentaje de error. Las frecuencias de reloj que minimizan este error a un 0.0 % para una generación de 9.600 baudios, son: 1.8432 Mhz, 3.6864 MHz, 7.3728 MHz, 11.0592 MHz, 14.7456 MHz y 18.4320 MHz. Se ha escogido la velocidad de 3.6864 Mhz como frecuencia de reloj a introducir en el circuito final para el microcontrolador AVR, dado que es una frecuencia que se puede encontrar fácilmente en el mercado y ofrece un consumo óptimo (a mayor frecuencia mayor consumo).

Configuración de los pines

El encapsulado del microcontrolador es de tipo DIP de 40 pines, como muestra la figura 5.4. Los elementos mínimos que han de estar conectados al microcontrolador son la alimentación y el reloj de cuarzo externo. La alimentación se obtiene del regulador de voltaje con tensión de +5V que existe ya en el circuito (para alimentar el módulo Bluetooth).

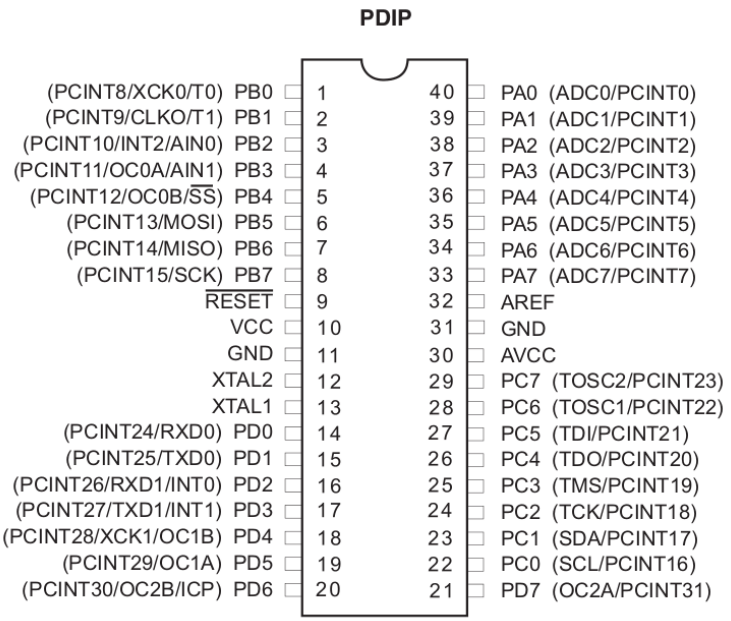


Figura 5.4: Esquema de los pines del microcontrolador AVR ATMega644A.

Además de los elementos mínimos mencionados, el microcontrolador ha de

tener las siguientes conexiones:

- Un conector AVR ISP (*In System Prgrammer*) para poder reprogramar el microcontrolador sin tener que extraerlo del circuito.
- Un botón de *reset* para permitir resetear el microcontrolador de forma manual.
- USART0 y USART1: la primera conecta el microcontrolador al interfaz ADS (en la práctica al integrado MAX232), y la segunda conecta el microcontrolador al módem Bluetooth.

Según las conexiones anteriores, la configuración final de los pines del microcontrolador es la siguiente:

- **Pin 6 - MOSI**: pin dedicado al conector AVR ISP.
- **Pin 7 - MISO**: pin dedicado al conector AVR ISP.
- **Pin 8 - SCK**: pin dedicado al conector AVR ISP.
- **Pin 9 - RESET**: pin conectado al botón de reset y al conector AVR ISP.
- **Pin 10 - VCC**: pin de alimentación con tensión de +5V.
- **Pin 11 - GND**: pin conectado a tierra.
- **Pin 12 - XTAL2**: pin conectado a una de las patillas del cristal de cuarzo.
- **Pin 13 - XTAL1**: pin conectado a una de las patillas del cristal de cuarzo.
- **Pin 14 - RXD0**: pin de RX de la USART0, conectado al pin TX del integrado MAX232.
- **Pin 15 - TXD0**: pin de TX de la USART0, conectado al pin RX del integrado MAX232.
- **Pin 16 - RXD1**: pin RX de la USART1, conectado al pin TX del módem Bluetooth.
- **Pin 17 - TXD1**: pin TX de la USART1, conectado al pin RX del módem Bluetooth.
- **Pin 30 - AVCC**: pin de alimentación con tensión de +5V.
- **Pin 31 - GND**: pin conectado a tierra.

5.2.4. Microcontrolador II - Software

El microcontrolador en el presente proyecto se limita a funcionar como una tubería o *pipe* entre el módem Bluetooth y el interfaz ADS, aunque en un futuro se ampliará su funcionalidad. Para cumplir con el objetivo del proyecto, el código debe habilitar las USARTS y leer bytes del módem Bluetooth y enviárselos al

interfaz ADS.

Diagrama de bloques

El diagrama de bloques de la figura 5.5 refleja el funcionamiento del código que llevará programado el microcontrolador. Cuando el circuito esté conectado al automóvil, el AVR tendrá alimentación y comenzará a ejecutar el código de manera infinita. Los pasos que realiza dicho código son los siguientes: al arrancar el microcontrolador, configura los registros y después se queda a la espera de recibir un byte del módem Bluetooth a través de la USART1; cuando se detecta la llegada de un byte por dicha USART, lo recepciona y lo envía por la USART0, conectada al interfaz ADS.

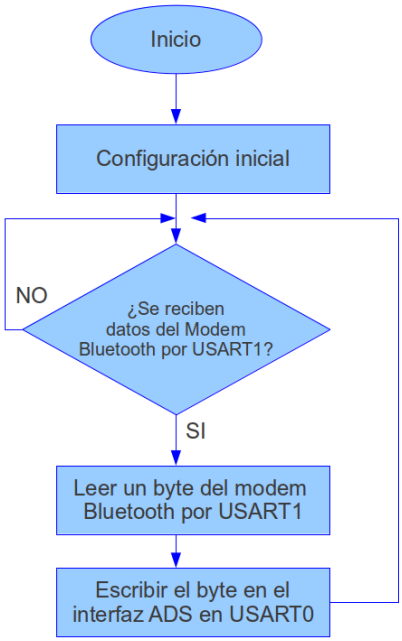


Figura 5.5: Diagrama de bloques del software del microcontrolador.

Configuración de los registros

Las dos USARTS del ATmega644a (USART0 y USART1) se han de configurar con los siguientes parámetros de conexión:

- 9.600 baudios
- 8 bits de datos
- Paridad par (*even*)
- 2 bit de stop

El microcontrolador tiene varios registros para la configuración de las USARTS, estos son: UDRn, UCSRnA, UCSRnB, UCSRnC, UBRRnL y UBRRnH. la letra "n" que aparece en los nombres anteriores, está referenciando a dos registros que existen en ambas USARTs, pero sustituyendo la "n" por 0 para la USART0 y 1 para la USART1. Por ejemplo, el registro UDR0 estaría en la USART0, y el registro UDR1 en la USART1.

A continuación se realiza una descripción de cada uno de los registros anteriores y de los bits que los integran, así como los valores con los que se van a configurar.

UDRn – USART *I/O Data Register* n

Cada USART contiene dos *buffer*s para las transferencias de datos: un *buffer* de transmisión donde se cargan los datos a transmitir, y un *buffer* de recepción, donde se almacenan los datos recibidos. Mediante el registro UDRn se accede a ambos *buffer*s, al *buffer* de recepción en las operaciones de lectura (RXB), y al *buffer* de transmisión, en las operaciones de escritura (TXB).

Si se configurasen 9 bits de datos, se utilizarían los bits RXB8n y TXB8n del registro UCSRnB (USART *Control and Status Register* n B) para almacenar el noveno bit de datos. En éste caso sólo son necesarios 8 bits de datos, por tanto, con los registros UDRn es suficiente.

UCSRnA – USART *Control and Status Register* A

Este registro de control está formado por los siguientes bits:

- **Bit 7 – RXCn**: USART *Receive Complete*, es un bit de *flag*, su valor es "uno" cuando hay datos sin leer en el *buffer* de recepción y pasa a "0" cuando se produce la lectura. También se utiliza para activar la interrupción de recepción de datos si está última está activada.
- **Bit 6 – TXCn**: USART *Transmit Complete*, es un bit de *flag*, su valor es "1" cuando los datos presentes en el *buffer* de envío han sido emitidos. También se utiliza para activar la interrupción de transmisión de datos, si esta última está activada.
- **Bit 5 – UDREn**: USART *Data Register Empty*, es un bit de *flag*, su valor es "1" cuando el *buffer* de transmisión (UDRn TXB) esta vacío, y por tanto, listo para recibir datos a enviar. También puede generar una interrupción cuando cuando el *flag* vale "1" (bit UDRIEn, ver página 98).
- **Bit 4 – FEn**: *Frame Error*, este bit es puesto a "1" si el siguiente carácter en el *buffer* de recepción tuvo un error de frame en la recepción. Esto es, cuando el primer bit de stop del siguiente carácter en el *buffer* de recepción es cero en lugar de uno.

- **Bit 3 – DORn**: *Data OverRun*, este bit es puesto a "1" cuando una condición de sobreescritura de datos es detectada. Ésta condición se da cuando el *buffer* de recepción de datos está lleno y se siguen recibiendo datos.
- **Bit 2 – UPEn**: *USART Parity Error*, este bit es puesto a "1" al detectarse un error de paridad en el *buffer* de recepción cuando están habilitados la recepción del bit de paridad y su comprobación (bits UPMn1 y UPMn0 del registro UCSRnC, página 99).
- **Bit 1 – U2Xn**: *Double the USART Transmission Speed*, este bit sólo se utiliza en las transmisiones asíncronas, si se pone a "1" dobla la velocidad de transmisión de datos, pero es más sensible a los errores de transmisión.
- **Bit 0 – MPCMn**: *Multi-processor Communication Mode*, este bit activa el modo de comunicación Multi-procesador. Cuando se activa éste modo, los datos recibidos sólo son tenidos en cuenta si tienen información de dirección, de lo contrario se desechan.

Los bits que se van a configurar de éste registro son U2Xn y MPCMn. En el caso de U2Xn, no interesa doblar la velocidad para no aumentar la sensibilidad a los errores de transmisión, por tanto, se pondrá el bit a "0". En el caso de MPCMn, no se va a utilizar el modo de multiprocesador, y el bit se pondrá también a "0".

UCSRnB – USART *Control and Status Register* n B

Este registro de control está formado por los siguientes bits:

- **Bit 7 – RXCIEn**: RX *Complete Interrupt Enable* n, este bit se utiliza para activar la interrupción de recepción de datos completada.
- **Bit 6 – TXCIEn**: TX *Complete Interrupt Enable* n, este bit se utiliza para activar la interrupción de transmisión de datos completada.
- **Bit 5 – UDRIEn**: USART *Data Register Empty Interrupt Enable* n, este bit se utiliza para activar la interrupción de *buffer* de recepción vacío y listo para enviar.
- **Bit 4 – RXENn**: *Receiver Enable* n, con el bit a "1" habilita la recepción de datos.
- **Bit 3 – TXENn**: *Transmitter Enable* n, con el bit a "1" habilita la transmisión de datos.
- **Bit 2 – UCSZn2**: *Character Size* n, éste bit combinados con los bits UCSZn1 y UCSZn0 del registro UCSRnC (ver página 99), determinan el número de bits de datos que se envían y reciben en cada frame.
- **Bit 1 – RXB8n**: *Receive Data Bit* 8 n, es el noveno bit del carácter recibido cuando se opera con frames de 9 bits de datos.
- **Bit 0 – TXB8n**: *Transmit Data Bit* 8 n, es el noveno bit del carácter

enviado cuando se opera con frames de 9 bits de datos.

Los bits que se van a configurar de éste registro son RXENn, TXENn y UCSZn2. RXENn se pondrá a "1" para habilitar la recepción de datos; TXENn se pondrá a "1" para habilitar la transmisión de datos, y por último, UCSZn2 se pondrá a "0" para establecer 8 bits de datos en las transmisiones.

UCSRnC – *USART Control and Status Register* n C

Este registro de control está formado por los siguientes bits:

- **Bits 7 y 6 – UMSELn1 y UMSELn0**: USART *Mode Select*. Estos dos bits indican si la USART va a funcionar en modo síncrono, asíncrono o Máster SPI. La tabla 5.1 muestra los valores de los bits para cada modo.

UMSELn1	UMSELn0	Modo
0	0	USART asíncrona
0	1	USART síncrona
1	0	(Reservado)
1	1	Master SPI (MSPIM)

Cuadro 5.1: Configuración de los bits UMSELn

- **Bits 5 y 4 – UPMn1 y UPMn0**: USART *Parity Mode*. Estos dos bits indican el tipo de paridad a utilizar en cada frame, es decir, si se usa o no el bit de paridad, y en caso afirmativo, si el bit indica paridad par o impar.La tabla 5.2 muestra los valores de los bits UPMn1 y UPMn0 para cada modo de paridad.

UPMn1	UPMn0	Modo de paridad
0	0	Deshabilitada
0	1	(Reservado)
1	0	Activado con paridad par (*even*)
1	1	Activado con paridad impar (*odd*)

Cuadro 5.2: Configuración de los bits UPMn

- **Bit 3 – USBSn**: USART *Stop Bit Select*. Éste bit indica si habrá uno o dos bits de stop en cada frame. La tabla 5.3 muestra los valores del bit USBSn según el número de bits de stop a utilizar.

- **Bits 2 y 1 – UCSZn1 y UCSZn0**: USART *Character Size*. Estos dos bits junto con el bit UCSZn2 del registro UCSRnC (ver página 98), indican el tamaño del carácter a enviar en cada frame. La tabla 5.4 refleja los valores de estos tres bits para configurar el tamaño del carácter. (5, 6, 7, 8 o 9 bits).

USBSn	**Bits de parada**
0	1-bit
1	2-bit

Cuadro 5.3: Configuración de los bits USBSn, Bits de parada

UCSZn2	**UCSZn1**	**UCSZn0**	**Tamaño del carácter**
0	0	0	5-bits
0	0	1	6-bits
0	1	0	7-bits
0	1	1	8-bits
1	0	0	(Reservado)
1	0	1	(Reservado)
1	1	0	(Reservado)
1	1	1	9-bit

Cuadro 5.4: Configuración de los bits UCSZn

- **Bit 0 – UCPOLn**: USART *Clock Polarity*, éste bit se utiliza solo en el modo síncrono de la USART. Sirve para establecer el modo de operación entre los datos de entrada y salida y la señal de reloj síncrono. En modo asíncrono no se utiliza éste bit.

Se utilizarán todos los bits del registro UCSRnC salvo el bit 0 (UCPOLn). Dado que las USARTS deben operar utilizando los siguientes parámetros:

- Modo asíncrono.
- Paridad par.
- 2 Bits de Stop.
- 8 Bits de datos (tamaño del carácter).

Los bits del registro quedarán configurados según la tabla 5.5.

Bit	0	1	2	3	4	5	6	7
Nombre	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn
Valor	0	0	1	0	1	1	1	0

Cuadro 5.5: Configuración de los bits del registro UCSRnC

UBRRnL y UBRRnH – USART *Baud Rate Registers*

Los registros UBRRnL y UBRRnH se utilizan para indicar la velocidad de las USARTS en baudios por segundo. El *datasheet* del microcontrolador incluye unas tablas indicando el valor de éstos registros según la frecuencia del oscilador del microcontrolador (velocidad del reloj), si se ha seleccionado duplicar o no la velocidad de transmisión (bit U2Xn), y la velocidad en baudios con la que configurar las USARTs.

Como la velocidad del reloj es de 3.6864 MHz (ver 5.2.3 en página 94), y se necesita que las USARTs trabajen a 9.600 baudios por segundo sin duplicar la tasa de transferencia (bit U2Xn a 0), los registros UBRRnH y UBRRnL han de tener los valores "0" y "23" respectivamente según el *datasheet*.

5.2.5. PCB (*Printed Circuit Board*)

El diseño del circuito impreso (PCB) consta de dos partes: un diseño esquemático y un diseño de placa. Ambos diseños se han realizado utilizando el software Eagle 6.2 de Cadsoft.

La figura 5.6 muestra el diseño esquemático. En él aparecen todos los elementos del circuito identificados junto con sus valencias en los casos que corresponde.

Respecto al diseño de la placa, la herramienta Eagle ha realizado automáticamente el proceso de creación de pistas. El diseño final de una sola cara se puede observar en la figura 5.7.

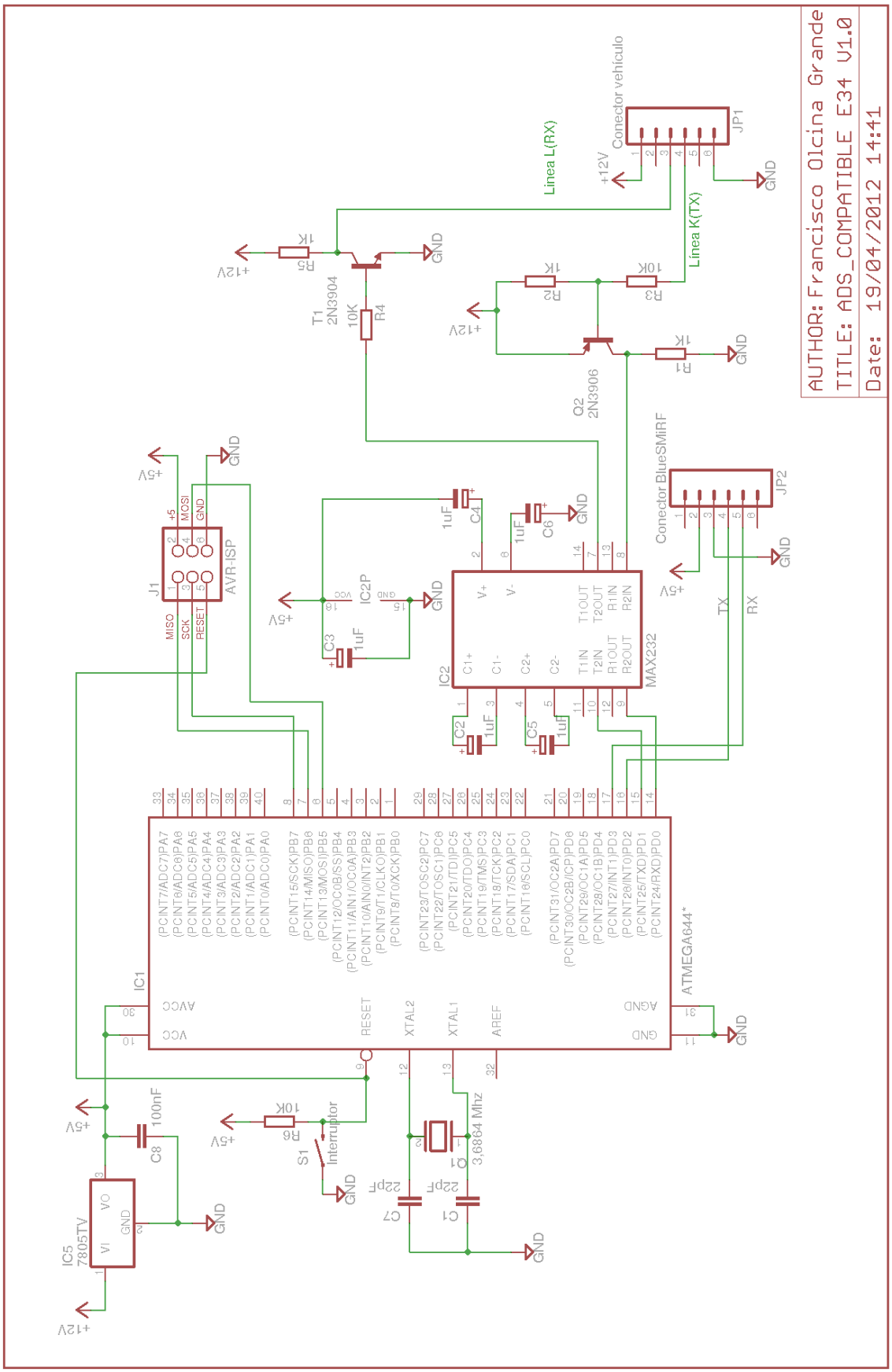


Figura 5.6: Diseño esquemático del circuito final.

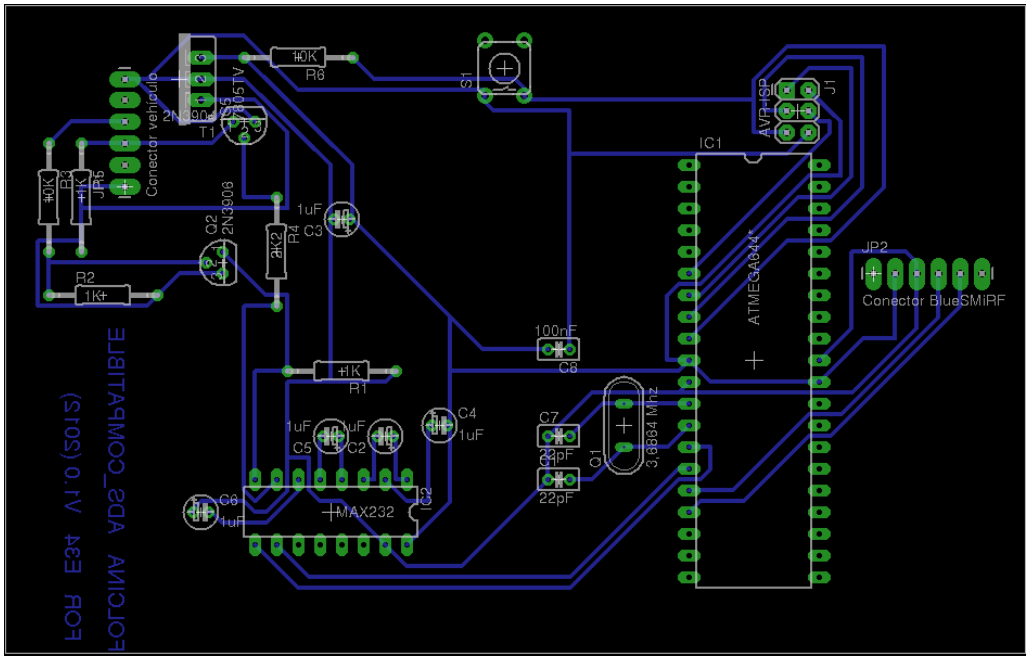


Figura 5.7: Diseño de la placa del circuito.

5.3. Implementación

En esta sección se detallan las fases para la fabricación y posterior configuración del circuito impreso, así como las pruebas para confirmar su correcto funcionamiento.

5.3.1. Fabricación del PCB

El circuito impreso (PCB) se ha fabricado de forma manual. En la fabricación profesional se suelen utilizar máquinas que realizan todo el proceso de forma automática, sin embargo el coste de fabricación de una sola placa como la del presente proyecto es elevado, y se ha optado por fabricar el PCB con los medios más económicos posibles, dando como resultado una placa completamente funcional.

La lista de componentes para la fabricación de la placa son los siguientes:

- 1 placa de cobre de al menos 12 x 8 cm.
- 1 regulador de voltaje LM7805.
- 1 cristal de cuarzo de 3,6864 Mhz.
- 1 integrado MAX232.
- 1 AVR ATMega644A de 40 pines con encapsulado DIP.

- 1 zócalo de 40 pines DIP.
- 1 zócalo de 16 pines DIP.
- 1 conector molex de 6 pines hembra.
- 1 conector molex de 6 pines macho.
- 1 conector de 6 pines hembra (2 filas de 3 pines).
- 1 interruptor.
- 2 condensadores cerámicos de 22 pF.
- 1 condensador cerámico de 100 nF.
- 5 condensadores electrolíticos de 1μF.
- 3 resistencias 1 KΩ.
- 2 resistencias 10 KΩ.
- 1 transistor NPN 2N3904.
- 1 transistor PNP 2N3906.
- 1 Módem BlueSMiRF Silver.

Primer paso - Impresión sobre la placa de cobre

Una vez obtenidos los componentes anteriores, el primer paso es imprimir el diseño de las pistas del circuito sobre la placa de cobre. Para ello se ha utilizado el método de transferencia de tóner que consiste en lo siguiente:

Se imprime en papel de transparencia el diseño de las pistas con una impresora láser. En las opciones de impresión ha de seleccionarse el modo de mayor calidad, para asegurar la mayor transferencia de tóner a la transparencia.

Después se prepara la placa de cobre limpiándola con alcohol y una lana de acero o similar (ver figura 5.8). La limpieza hay que hacerla en círculos, rayando superficialmente la placa de cobre para ayudar a que el tóner se agarre mejor.

Con la transparencia impresa y la placa limpia y preparada, lo siguiente que se necesita es una plancha (ver figura 5.9); se utilizará el calor que genera la plancha para transferir el tóner de la transparencia impresa a la superficie de la placa. Para ello, se cortará la transparencia para que ocupe la superficie de la placa y después se pondrá sobre ésta última, pegada con cinta transparente para que no se mueva.

Posteriormente se coloca un trapo encima de la placa con la transparencia pegada en su parte superior, y se pasa la plancha caliente durante varios minutos por encima del trapo (ver figura 5.10). En este proceso la plancha calienta la transparencia provocando que tóner se desprenda y se adhiera a la placa.

Figura 5.8: Limpieza de la placa de cobre.

Figura 5.9: Materiales para la transferencia de tóner mediante calor.

Figura 5.10: Transferencia del tóner mediante calor.

Una vez pasada la plancha varios minutos, se retira el trapo, y se sumerge la placa con la transparencia pegada en agua fría. Hay que tener cuidado en este proceso dado que la placa esta a una temperatura elevada y puede quemar. Después de dos o tres minutos, la placa estará de nuevo fría y se podrá retirar la transparencia sin levantar el tóner de la placa. En este punto se tiene la placa con el dibujo de las pistas sobre el cobre.

Segundo paso - atacado del cobre

El segundo paso consiste en eliminar el cobre sobrante de la placa, dejando sólo el cobre que dibuja las pistas del circuito. Para ello se ha de introducir la placa en un ácido especial que ira eliminando el cobre de la misma salvo la parte protegida por el tóner de la impresora, que es el dibujo de las pistas.

El ácido se obtiene a partir de una mezcla de 2 partes de agua fuerte (ácido clorhídrico o HCI) con 4 partes de agua oxigenada (peróxido de hidrógeno o H_2O_2). En la figura 5.11 se observan los materiales para proceder al atacado del cobre: una cubeta donde alojar la placa con el ácido, unos guantes y unas gafas de protección, la placa con el tóner adherido dibujando las pistas, y una botella de aguafuerte y otra de agua oxigenada.

Al sumergir la placa de cobre sobre el ácido en la cubeta, éste tiene mantiene aun su color transparente (ver figura 5.12), pero en pocos segundos el ácido comienza una reacción química con el cobre de la placa que torna el líquido en un color más verdoso. Después de unos 45 minutos el cobre sobrante no cubierto por el tóner ha desaparecido casi en su totalidad, como se puede apreciar en la figura 5.13. Los últimos minutos son de especial importancia porque hay que retirar la placa de la cubeta cuando haya desaparecido todo el cobre sobrante pero antes de que comience el ácido a eliminar el cobre que está debajo del tóner.

Figura 5.11: Materiales para el atacado del cobre.

Al retirar la placa de la cubeta, se lava con agua abundante para eliminar todos los restos de ácido. Después hay que limpiar la superficie con alcohol para eliminar los restos de tóner y que el cobre que dibuja las pistas sea visible en la superficie.

Figura 5.12: Comienzo del atacado del cobre.

Tercer paso - perforado de la placa

El tercer paso consiste en perforar la placa en aquellas partes donde se vayan a insertar los distintos componentes. Para ello se requiere un punzón, un martillo, un par de soportes para sujetar la placa por los lados y una máquina para hacer taladros con brocas pequeñas (una herramienta *dremel* o similar).

Primero se procede al marcado de los agujeros. Para ello se coloca la placa sobre los soportes, dejando el centro de la misma al aire, como se observa en la

Figura 5.13: Atacado del cobre después de 45 minutos de reacción química.

figura 5.14. Con el punzón y el martillo se hace una pequeña marca en cada agujero que haya que realizar con el taladro. Ésta marca facilita que la broca del taladro no patine mientras realiza la perforación.

Una vez hechas todas las marcas sobre los agujeros a realizar en la placa, se procede a taladrarlos con la medida de broca correspondiente según el diámetro del agujero a realizar. Las medidas del diámetro de las brocas van desde 0,7 mm hasta 1 mm.

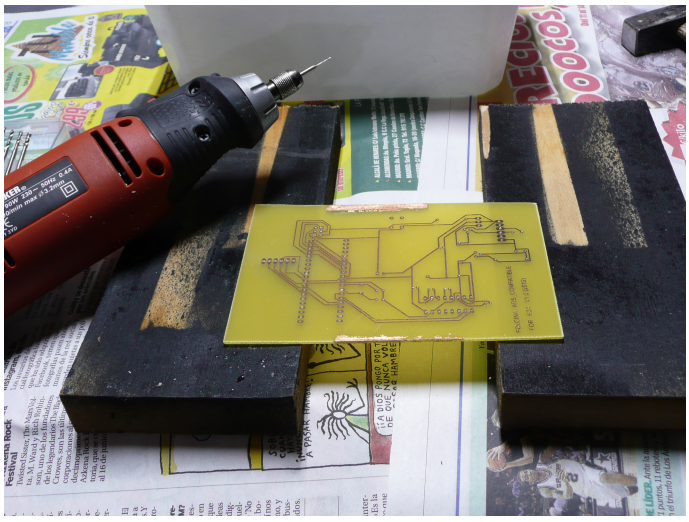


Figura 5.14: Perforado de la placa.

Cuarto paso - Soldado de componentes

El cuarto paso consiste en soldar todos los componentes a la placa, para ello se necesita un soldador, estaño, pasta para soldar, un soporte multibrazo para sujetar la placa, y una herramienta de corte.

Partiendo de la placa con las perforaciones hechas, se van situando en primer lugar los componentes más grandes introduciendo sus patillas sobre los agujeros de la cara sin cobre, soldando las mismas por la cara donde está el cobre. La figura 5.15 muestra el montaje superficial de los zócalos y los conectores sobre el lado sin cobre de la placa; el lado con las pistas de cobre también aparece a trasluz. Una vez realizado el montaje superficial, sin que los componentes se muevan de su sitio, se sueldan con estaño y pasta de soldar las patillas al cobre.

Una vez soldados los zócalos y conectores, se pasan a soldar el resto de componentes: resistencias, transistores, condensadores, respetando en el caso de los dos últimos su posición y polaridad. Después de soldar estos componentes, es necesario cortar el sobrante de sus patillas con la herramienta de corte.

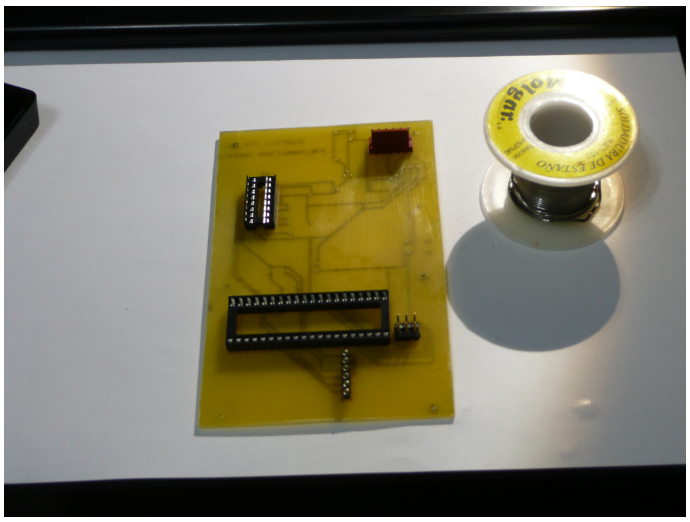


Figura 5.15: Montaje superficial de los componentes más grandes.

Quinto paso - comprobación de la placa

En este paso se alimentará la placa con una tensión de +12V y se verificarán los voltajes en varios puntos y en caso de ser correctos se procederá a insertar los integrados (el AVR ATMega644A y el MAX232) en sus correspondientes zócalos. Las herramientas necesarias son: un multímetro y una fuente de alimentación de corriente continua con tensión de +12V.

En primer lugar se comprobará que no hay cortes en las pistas, para ello se pondrá el multímetro en la posición de medición de continuidad y se comprobarán todos los puntos que estén conectados, cada uno con una de las dos puntas del multímetro. Ha de sonar un pitido cuando haya conectividad. Si en algún momento el multímetro no indica conectividad entre dos puntos que han de estar conectados, hay que revisar la pista que los une y detectar dónde está el corte. De existir dicho corte, se estañará sobre el mismo un pequeño alambre para dar continuidad a la pista.

En segundo lugar se comprobará que las tensiones del circuito son las correctas

en todos los puntos. Para ello se utilizará la fuente de alimentación con tensión de +12V de corriente continua sobre el conector de la placa que se utilizará para el vehículo. Éste conector tiene seis pines, de los cuales, el pin número uno obtendría tensión de +12V del vehículo, y el conector número seis obtendría tierra del vehículo. Para realizar la prueba, se conectará el borne positivo de la fuente al pin número uno del conector, y el borne negativo o de tierra al pin número seis. A partir de ese momento, la placa queda alimentada con tensión de +12V.

Con el multímetro en modo de comprobación de tensión continua, hay que comprobar los siguientes elementos:

- **Regulador de tensión LM7805**: el regulador tiene 3 patillas correspondientes a la tensión de entrada, tierra y tensión de salida. Se pondrá la punta negativa del multímetro sobre la patilla de tierra y con la punta positiva se comprobará la tensión de entrada y la de salida. Los valores que debe dar son +12V y +5V respectivamente.
- **Zócalo de 40 pines del AVR**: en el zócalo donde se insertará el AVR hay que comprobar los pines relativos a la alimentación del microcontrolador. Éstos pines son el número 10 y el número 30 para la alimentación positiva, y el número 11 y el 31 para la alimentación negativa. Se pondrá la punta negativa del multímetro sobre el pin número 11 y se comprobará con la punta positiva que los pines 10 y 30 dan una tensión de +5V. Después se repetirá el proceso poniendo la punta negativa sobre el pin 31.
- **Zócalo de 16 pines del MAX232**: en el zócalo donde se insertará el MAX232 hay que comprobar los pines relativos a la alimentación del integrado. Éstos pines son el número 16 para la alimentación positiva, y el número 15 para tierra. Se pondrá la punta negativa del multímetro sobre el pin número 15 y se comprobará con la punta positiva que el pin número 16 da una tensión de +5V.
- **Conector de 6 pines del módem Bluetooth**: en el conector donde se insertará el módem Bluetooth, los pines 2 y 3 son de alimentación, el pin número 2 para tensión positiva y el pin número 3 para tierra. Se pondrá la punta negativa del multímetro sobre el pin número 3 y se comprobará que el pin número 2 de una tensión de +5V.
- **Conector de 6 pines del AVR ISP**: en el conector a través del cual se programará el microcontrolador AVR, los pines 2 y 6 corresponden a la alimentación, siendo el pin número 2 de alimentación positiva y el pin número 6 de tierra. Se pondrá la punta negativa del multímetro sobre el pin número 6 y se comprobará que el pin número 2 de una tensión de +5V.

Sexto paso - montaje de componentes superficiales

En este paso se han de montar los componentes superficiales de los que consta la placa y que van conectados a los zócalos y conectores existentes. Los componentes a

montar son: el microcontrolador, sobre el zócalo de 40 pines; el integrado MAX232, sobre el zócalo de 16 pines; y por último el módem Bluetooth, sobre el conector de 6 pines reservado al propio módem. La figura 5.16 muestra el aspecto del circuito una vez finalizado y con todos sus componentes montados.

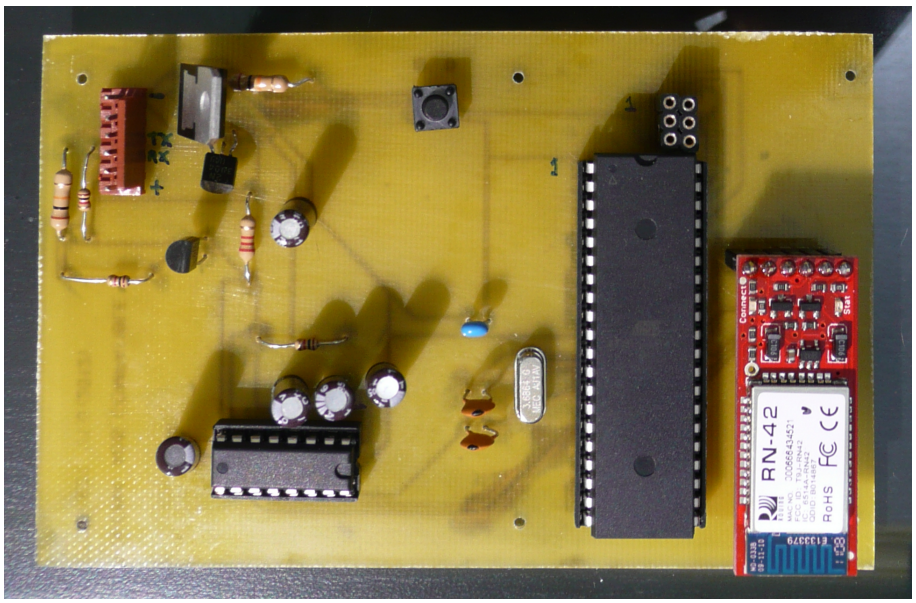


Figura 5.16: Circuito final con todos sus componentes montados.

5.3.2. Configuración del módem Bluetooth

El módem Bluetooth debe se ha configurado con los siguientes parámetros:

- Modo esclavo, donde otros dispositivos pueden detectarlo y conectarse a él.
- Puerto serie configurado a 9.600 baudios, 8 bits de datos, paridad par y 2 bits de stop.
- Contraseña de seguridad activada.

La configuración del módem puede hacerse vía Bluetooth o bien mediante una conexión física a los pines del módem. Dado que no se dispone de un transmisor Bluetooth en los ordenadores utilizados para el proyecto, se ha optado por realizar la configuración del módem utilizando una conexión física.

La conexión física se realiza conectando el puerto serie del ordenador (líneas RX y TX), al módem Bluetooth (pines RX y TX), no obstante, el módem funciona con el estándar TTL y el puerto serie del ordenador con el estándar RS-232C. Para adaptar un estándar a otro, se ha tenido que utilizar un circuito que incorpore el integrado MAX232, el cual transforma los voltajes de TTL a RS-232C y viceversa.

La figura 5.17 muestra el circuito utilizado. Dado que se disponía de una fuente de alimentación de +12V, se ha necesitado también un regulador de voltaje que

transforme la tensión de +12V a +5V (integrado LM7805), que es la que necesita el módem Bluetooth.

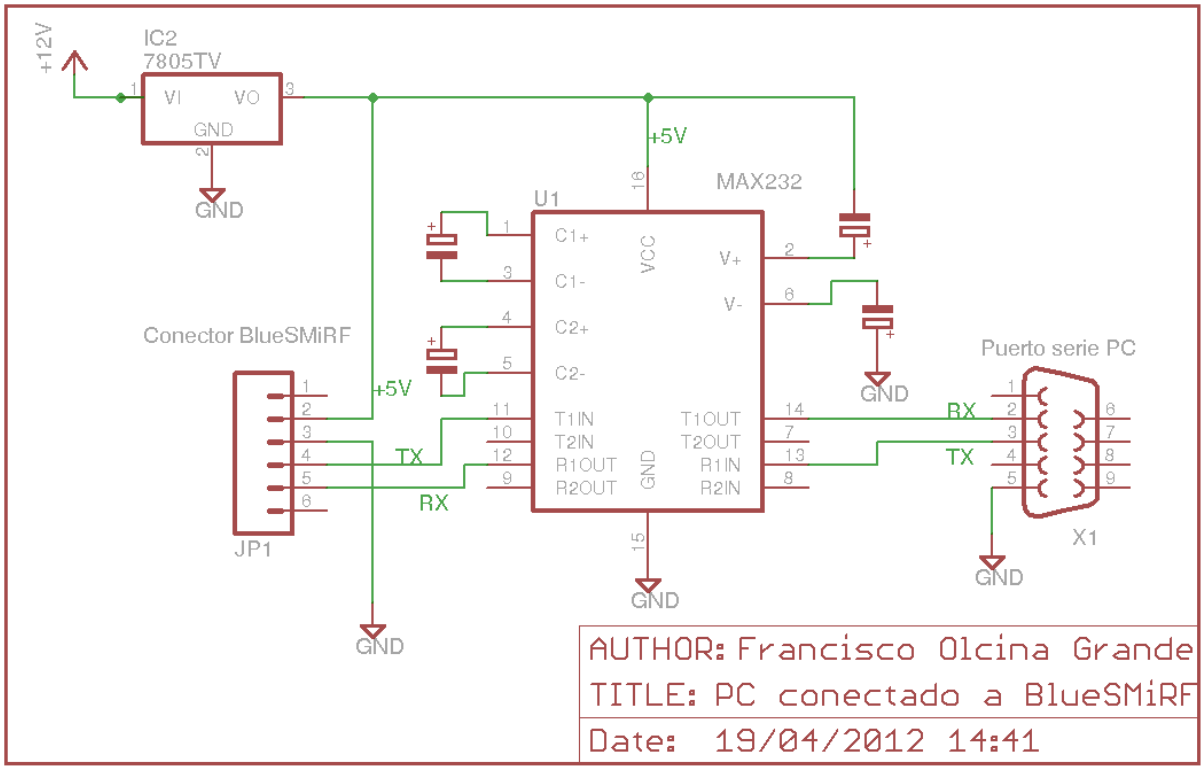


Figura 5.17: Circuito para la conexión del PC al módem Bluetooth.

Comandos de configuración

Una vez creado el circuito, se ha conectado el módem Bluetooth y el puerto serie COM1 del PC al mismo, y se ha abierto un terminal para poder realizar la comunicación con el módem. Los parámetros de conexión para el terminal han sido: 115.200 baudios, 8 bits de datos, sin paridad, 1 bit de stop.

Desde el terminal, se ha introducido lo siguiente para acceder al modo de comandos del módem:

```
$$$
```

Una vez accedido a dicho modo, se han ejecutado los siguientes comandos:

- **SA,1**: para activar la autenticación.
- **SE,1**: para activar el cifrado de datos.
- **SL,E**: para activar la paridad par.
- **SM,0**: para activar el modo esclavo.
- **SN,BMW-INTERFAZ**: configura el nombre del módem Bluetooth con el

valor "BMW-INTERFAZ".

- **SP,*XXXXXX***: configura el código PIN del módem, donde *XXXXXX* es el valor del código.
- **SQ,256**: configura el módem para utilizar 2 bits de stop.
- **ST,0**: deshabilita la configuración vía Bluetooth, sólo es posible por conexión física.
- **SU,9600**: configura la velocidad de conexión a 9.600 baudios.
- **S ,0**: activa el perfil de puerto serie o SPP (*Serial Port Profile*).
- **S|,0103**: activa el modo de ahorro de energía para los intervalos sin conexión. Realiza un ciclo en el que está 1 segundo encendido y esperando conexiones, y 3 segundos apagado.

Después de configurar el módem, se ha reiniciado para que utilice los nuevos valores mediante:

```
R,1
```

Una vez reiniciado el módem, se ha configurado de nuevo el terminal para utilizar los nuevos parámetros de conexión, es decir, 9.600 baudios, 8 bits de datos, paridad par y 2 bits de stop. Después se ha entrado de nuevo al modo comandos, y a continuación se ha ejecutado el comando que lista la información del módem. De ésta forma se ha podido verificar que opera con los nuevos parámetros, tal y como se muestra a continuación:

```
$$$
D
***Settings***
BTA=000666434521
BTName=BMW-INTERFAZ
Baudrt=9600
Parity=Even
Mode  =Slav
Authen=1
Encryp=1
PinCod=XXXXX
Bonded=0
Rem=NONE SET
```

Dirección MAC del módem Bluetooth

El parámetro BTA (Bluetooth *Address*) mostrado en el listado anterior se corresponde con la dirección MAC del módem Bluetooth. Se ha anotado para su posterior utilización en la aplicación a desarrollar sobre la plataforma Android. Su valor es:

```
00:06:66:43:45:21
```

5.3.3. Programación del microcontrolador

Para el desarrollo del código del microcontrolador se ha utilizado la herramienta AVR Studio 4.19 de ATMEL, disponible gratuitamente en la web de la compañía [4]. Se trata de un entorno de desarrollo (IDE) para microcontroladores de de 8 bits de ATMEL. Permite desarrollar el código en lenguaje ensamblador o en C/C++ y después programarlo en el microcontrolador utilizando un programador compatible.

El código fuente desarrollado para el microcontrolador del presente proyecto se ha realizado en lenguaje C, y está incluido en el anexo 9.2.6. Respecto al programador, se ha utilizado el AVRISP mkII, también de la compañía ATMEL.

A continuación se detallarán los pasos realizados para programar el código en el microcontrolador.

Conexión del programador

El programador AVRISP mkII debe conectarse a la placa a través del conector AVR ISP. Dado que la placa debe alimentarse con tensión de +12V, se ha utilizado la fuente de alimentación disponible en la mesa de trabajo.

La alimentación del circuito se ha obtenido a través del conector destinado al vehículo, concretamente de sus pines 1 y 6, (+12V y tierra respectivamente). Se ha conectado la fuente de alimentación a dichos pines para alimentar el circuito.

Con la placa alimentada, se ha conectado el programador al circuito (ver figura 5.18) y a un puerto USB del PC.

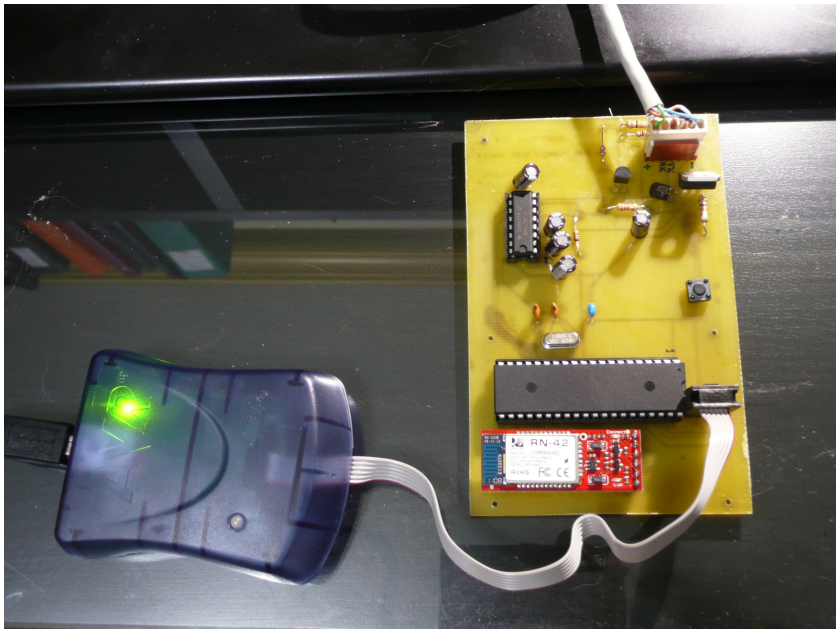


Figura 5.18: Programador AVRISP mkII conectado al circuito.

Programación mediante AVR Studio

En el PC, se ha abierto el proyecto que contiene el código desarrollado para el microcontrolador con la herramienta AVR Studio, y una vez abierto, se ha pulsado sobre el icono *Build Active Configuration*, como se puede observar en la figura 5.19. Este paso ha compilado el código escrito en C y ha generado un fichero binario con extensión .hex para programar el AVR.

Al compilar el programa, se ha generado un fichero binario con 1020 bytes de tamaño (aproximadamente 1 KByte). Dado que la memoria de código programable del microcontrolador es de 64KBytes, habrá espacio suficiente para programar el código generado.

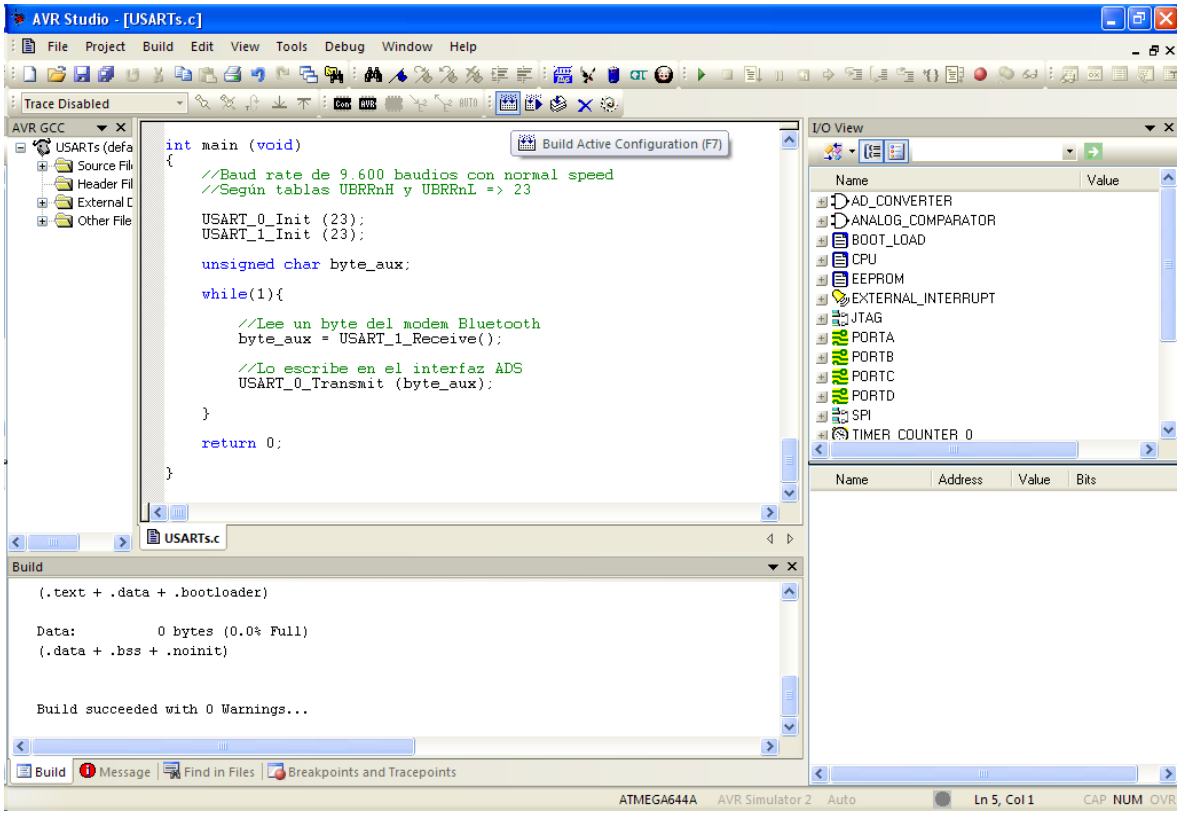


Figura 5.19: Compilación del código del microcontrolador mediante AVR Studio.

Después de la compilación, se ha pulsado sobre el icono *Connect to the Selected AVR Programmer*, como muestra la figura 5.20. Éste paso ha abrierto una ventana con las opciones de programación del microcontrolador utilizando el AVRISP mkII, como muestra la figura 5.21.

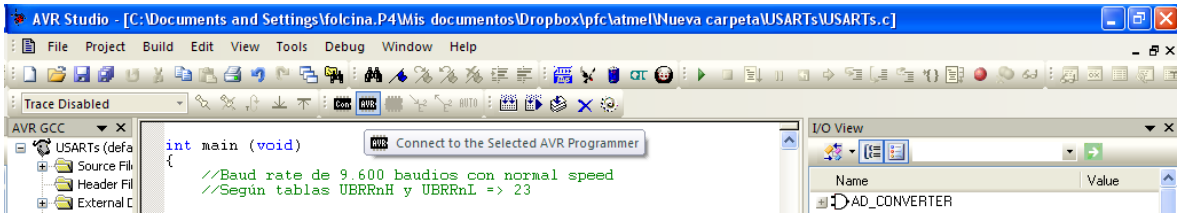


Figura 5.20: Conexión con el programador del microcontrolador en AVR Studio.

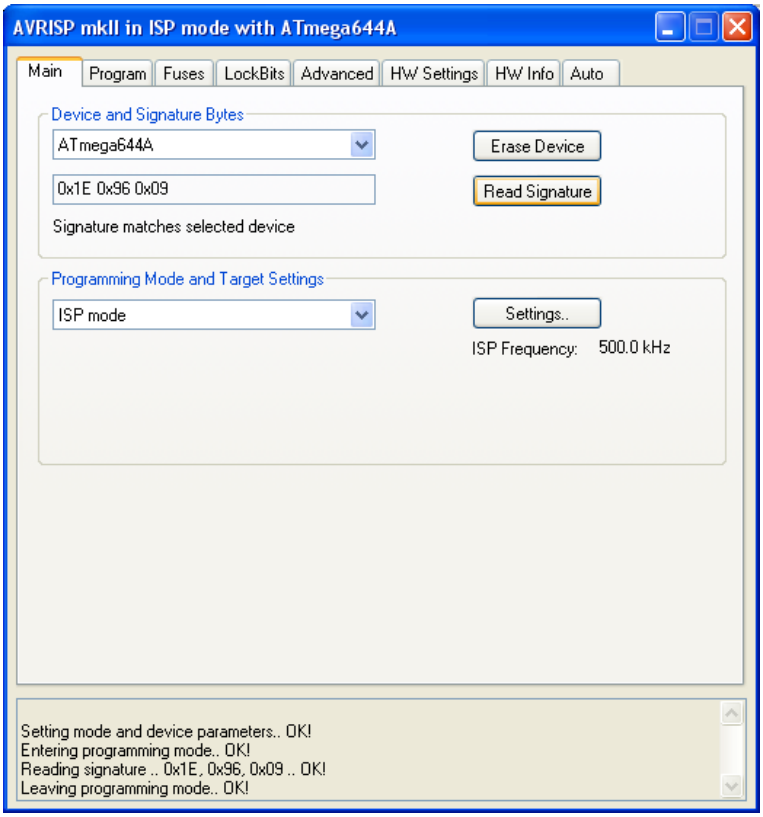


Figura 5.21: Selección del microcontrolador, del modo y de la velocidad de programación en AVR Studio.

En la ventana de la figura 5.21 aparece la pestaña *Main* de las opciones de programación. En ésta pestaña se ha seleccionado el microcontrolador, la velocidad, y el modo de programación.

El microcontrolador se ha seleccionado a través del menú desplegable superior que aparece en la sección *Device and Signature Bytes*, y en este caso se ha seleccioando el ATmega644A.

El modo de programación se ha seleccionado en el desplegable de la sección *Programming Mode and Target Settings*, y en esta caso ha sido el modo ISP.

La velocidad de programación debe ser debe ser aproximadamente 1/4 de la velocidad del cristal conectado al microcontrolador, en este caso 500 KHz (el cristal tiene una velocidad de 3,6864 MHz). Ésta velocidad se ha seleccionado a través del botón *Read Signature*.

Por último, una vez configurados los valores anteriores, se ha realizado la comprobación de la firma del dispositivo pulsando sobre *Read Signature*. Éste botón conecta el programador con el microcontrolador para verificar si la conexión es óptima y se puede realizar la programación. Cuando la operación se realiza con éxito, aparece el mensaje *Signature matches selected device*.

En la figura 5.22 aparece la configuración para los *Fuses* del microcontrolador. Éstos fuses establecen varias características de operación del AVR que sólo se pueden modificar en el momento de programarlo. Los *fuses* más relevantes son: OCDEN y JTAGEN, que permiten la depuración y deben estar deshabilitados para no consumir recursos del microcontrolador; SPIEN que debe estar habilitado para permitir la programación serial de bajo voltaje (+5V); y el SUT_CKSEL que indica que hay un cristal oscilador externo que funciona entre 3 y 8MHz, y que el retardo de arranque del microcontrolador debe ser de 256 ciclos de reloj y 65 ns (éste retardo sirve para que se estabilice la corriente antes de que el AVR comience a trabajar).

Una vez configurados los *fuses* se ha pulsado sobre *Program* grabandose los cambios al microcontrolador.

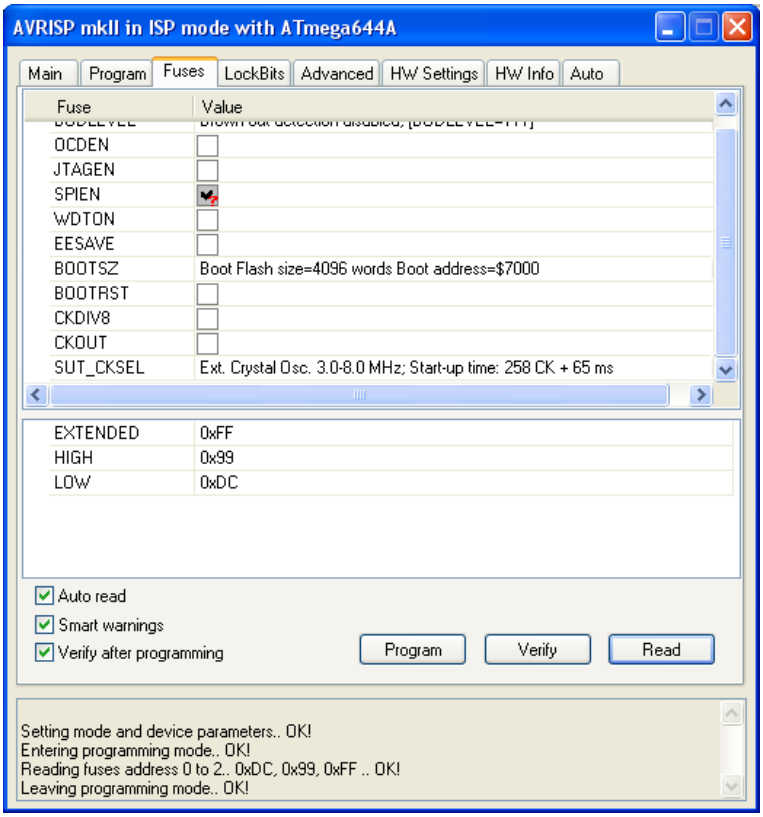


Figura 5.22: Programación de los *fuses* del microcontrolador en AVR Studio.

En la figura 5.23 aparece la pantalla de programación del código en el microcontrolador. En ella, se ha pulsado sobre *Input HEX file* para seleccionar el fichero .hex generado en la compilación del código C. Después, se ha pulsado en *Program* para programar el código en el microcontrolador, y por último, se ha reiniciado el AVR automáticamente y ha comenzando a ejecutar el código programado.

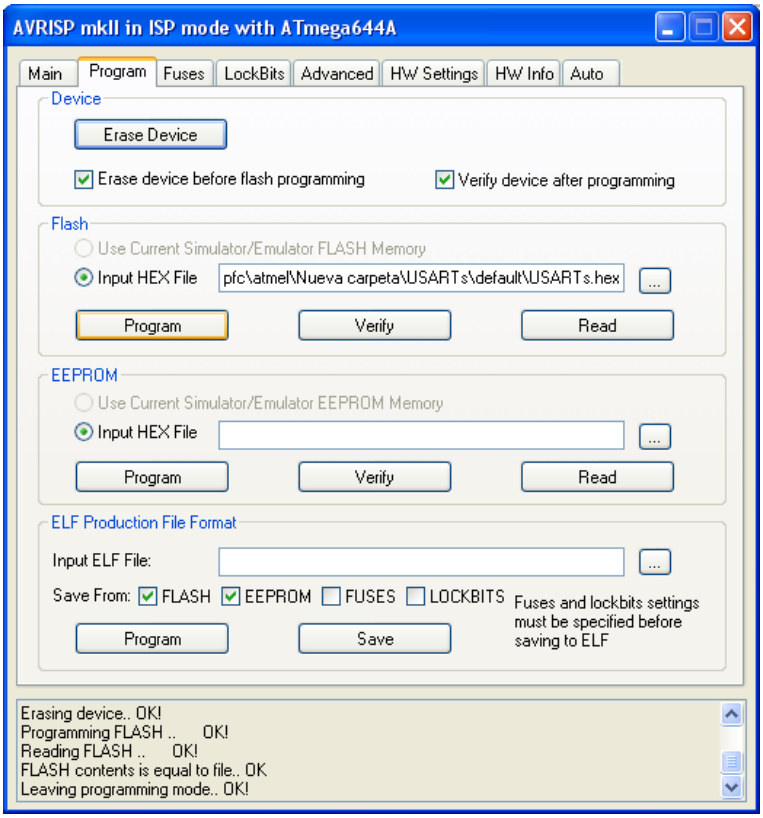


Figura 5.23: Programación del código en el microcontrolador desde AVR Studio.

5.3.4. Ensamblado del PCB

El último proceso que se ha realizado con el PCB es el de ensamblado. Se ha colocado dentro de una caja de plástico con las dimensiones adecuadas y con un conector externo. Se ha utilizado además un cable cuyos extremos se conectan por un lado al conector externo del PCB ensamblado y por otro lado al vehículo.

La lista de componentes que se han utilizado es la siguiente:

- 1 caja de plástico para PCB de 15 x 9 cm.
- 1 porta fusible aéreo.
- 1 fusible de 0,5 amperios.
- 1 conector serie DB9 macho.
- 1 conector Molex de 6 pines hembra.
- 1 taco de plástico con adhesivos en los extremos para fijar el módem Bluetooth.
- 1 cable con conector OBD I de 20 pines (de rosca) en un extremo y conector serie DB9 hembra en el otro extremo.

La caja de plástico utilizada para el ensamblaje del PCB tiene unas dimensiones de 15 x 9 x 3 cm, suficientes para alojar el PCB en su interior y con espacio sobrante para los cables que conectarán el PCB al conector externo de la caja.

La caja no contaba inicialmente con un conector externo, por lo tanto se ha añadido uno de tipo serie DB9 macho, compatible con el cable externo que unirá la caja al vehículo. Internamente se ha unido el conector DB9 con el PCB mediante cables, añadiendo un portafusibles aéreo en el cable que se encarga de la alimentación (+12V).

La tabla 5.6 muestra la correspondencia entre los pines del conector molex integrado en la placa y los pines del conector externo DB9 de la caja.

Pin del conector DB9 (caja)	**Pin del conector Molex (PCB)**	**Función**
2	6	GND
4	4	Linea TX (K)
8	3	Línea RX (L)
9	1	+12V

Cuadro 5.6: Correspondencia entre los pines del PCB y los del conector de la caja.

En el portafusibles se ha introducido un fusible de 0.5 amperios, suficiente para proteger todo el circuito. Para la elección de dicho valor se ha tenido en cuenta

que el AVR es el componente que requiere mayor intensidad de corriente de todo el circuito, con un máximo de 200 mA y aproximadamente 20 mA de media.

Para finalizar el proceso de ensamblado, se ha atornillado el PCB a la caja y una vez fijado, se ha unido con el conector externo. Adicionalmente, se ha colocado un taco de plástico con adhesivos en ambos lados para sujetar el módem Bluetooth al PCB. En la figura 5.24 se puede observar el PCB totalmente ensamblado.

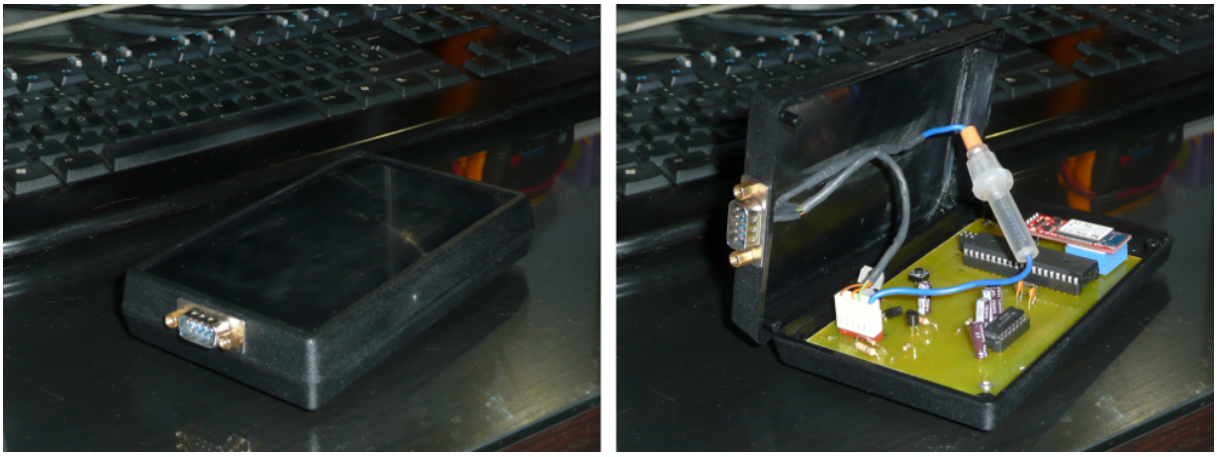


Figura 5.24: Interfaz de conexión remota.

Capítulo 6

Aplicación en Android

6.1. Introducción

En este capítulo se detalla el desarrollo de la aplicación para el S.O Android. El capítulo se divide en: una sección de diseño donde se muestra el diseño arquitectónico basado en componentes y los interfaces de usuario; y una sección de implementación, donde se muestra el código desarrollado y su implantación en el dispositivo móvil.

6.2. Diseño

En esta sección se muestra la fase de diseño de la aplicación, donde se detallan sus componentes, las clases que la implementan y por último las pantallas o interfaces de usuario.

6.2.1. Componentes del sistema

La aplicación se divide en 4 componentes principales. Éstos componentes se encuentran dentro del nodo físico denominado "Dispositivo Móvil" que representa el teléfono. La figura 6.1 muestra un diagrama de despliegue donde aparecen los componentes y los nodos físicos que representan el teléfono y el interfaz de conexión remota.

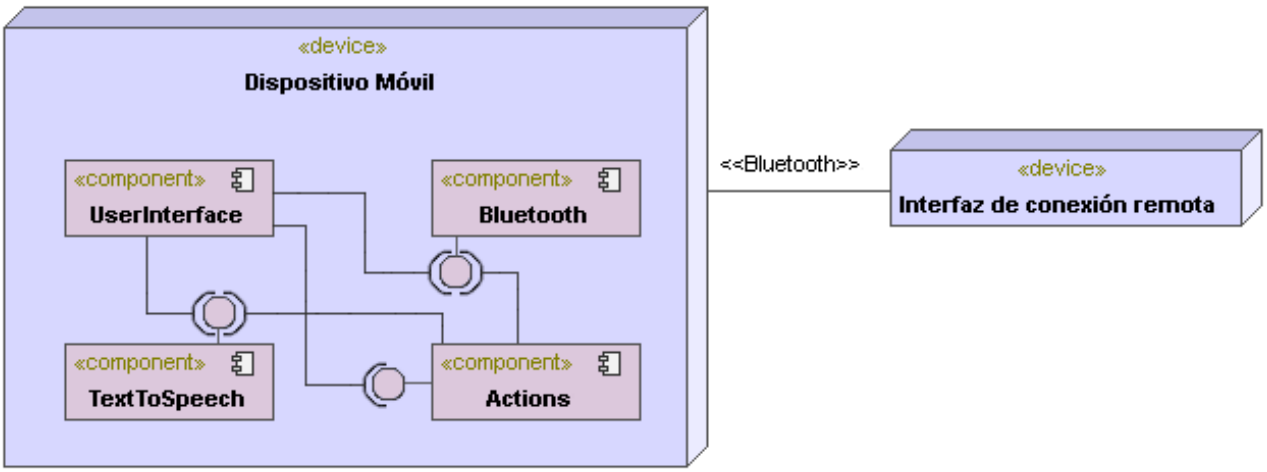


Figura 6.1: Diagrama de despliegue de la aplicación.

A continuación se listan las especificaciones de cada componente siguiendo un formato tabular que contendrá la siguiente información:

- **Identificador**: es un identificador único para cada componente con la forma COMP-X, donde X representa un número decimal que sirve para enumerar de forma secuencial los componentes.
- **Nombre**: define un nombre para cada componente.
- **Propósito**: detalla el propósito del componente en la aplicación.
- **Funciones**: lista las funciones del componente dentro de la aplicación.
- **Referencias**: lista los requisitos de usuario que originan la existencia del componente.

Identificador	COMP-1
Nombre	UserInterface
Propósito	Éste componente del sistema permite mostrar las distintas interfaces de usuario con las que cuenta la aplicación.
Funciones	▪ Mostrar interfaz de usuario principal: ésta interfaz consta de un botón de reconocimiento de voz para captar las órdenes emitidas por el usuario y ejecutarlas. También cuenta con otro botón para mostrar el interfaz de usuario con el listado de acciones. ▪ Mostrar interfaz de usuario con el listado de acciones: muestra todas las acciones disponibles y permite ejecutarlas al pulsar sobre cualquiera de ellas. ▪ Mostrar interfaz de usuario con los datos sobre el autor. ▪ Mostrar interfaz de usuario con la ayuda. ▪ Mostrar textos informando de las acciones realizadas o los errores que se produzcan.
Referencias	RUC-01, RUC-02, RUC-03, RUC-04, RUC-05, RUC-06, RUC-07, RUC-08, RUC-09, RUC-10, RUC-11, RUC-13, RUR-01, RUR-02, RUR-03, RUR-05

Cuadro 6.1: Especificación del componente COMP-1.

Identificador	COMP-2
Nombre	TextToSpeech
Propósito	Éste componente del sistema tiene como propósito realizar las funciones de síntesis de voz que utilizan las distintas partes de la aplicación
Funciones	▪ Expresar con voz la operación a realizar ▪ Expresar con voz cualquier tipo de error.
Referencias	RUC-12, RUC-13, RUR-01, RUR-03

Cuadro 6.2: Especificación del componente COMP-2.

Identificador	COMP-3
Nombre	Actions
Propósito	Éste componente del sistema tiene como propósito ejecutar las operaciones que se pueden realizar sobre el vehículo.
Funciones	▪ Abrir y cerrar las ventanas. ▪ Abrir y cerrar (con cierre normal o doble) el vehículo. ▪ Abrir y cerrar el techo eléctrico. ▪ Activar el limpiaparabrisas.
Referencias	RUC-01, RUC-02, RUC-03, RUC-04, RUC-05, RUC-06, RUC-07, RUC-08, RUC-11, RUC-12, RUC-13, RUR-01, RUR-03

Cuadro 6.3: Especificación del componente COMP-3.

Identificador	COMP-4
Nombre	Bluetooth
Propósito	Éste componente del sistema tiene como propósito realizar las funciones de comunicación Bluetooth entre el móvil y el interfaz de conexión remoto.
Funciones	▪ Conectar con el módem Bluetooth del interfaz remoto instalado en el vehículo. ▪ Enviar datos desde el móvil al módem Bluetooth del interfaz remoto.
Referencias	RUR-01, RUR-04

Cuadro 6.4: Especificación del componente COMP-4.

Por último, se ha construido una matriz de trazabilidad con objeto de verificar si el conjunto de componentes de la fase de diseño cumple todos los requisitos de usuario obtenidos en la fase de análisis. En la matriz de la tabla 6.5 se puede observar de una manera ágil y sencilla que todos los requisitos están cubiertos por los componentes del sistema.

Componente / **Requisito usuario**	**COMP-1**	**COMP-2**	**COMP-3**	**COMP-4**
RUC-01	X		X	
RUC-02	X		X	
RUC-03	X		X	
RUC-04	X		X	
RUC-05	X		X	
RUC-06	X		X	
RUC-07	X		X	
RUC-08	X		X	
RUC-09	X			
RUC-10	X			
RUC-11	X		X	
RUC-12		X	X	
RUC-13	X	X	X	
RUR-01	X	X	X	X
RUR-02	X			
RUR-03	X	X	X	
RUR-04				X
RUR-05	X			

Cuadro 6.5: Matriz de trazabilidad.

6.2.2. Clases

En esta sección se muestran las clases que van implementan la aplicación. La figura 6.2 muestra el diagrama de clases, donde se pueden observar las relaciones entre las mismas, así como los componentes dónde se encuadran.

El componente *UserInterface* engloba las clases que implementan las pantallas o interfaces de usuario, éstas clases son:

- ***MainActivity***: implementa la pantalla de usuario principal.
- ***ActionsListActivity***: implementa la pantalla de usuario con la lista de acciones posibles.
- ***About***: implementa la pantalla de usuario que muestra información sobre el autor.
- ***Help***: implementa la pantalla de usuario que muestra la ayuda de la aplicación.

Éstas clases implementan un *Activity* por lo tanto tendrán al menos un método *onCreate()*.

El componente *TextToSpeech* contiene la clase *MyTextToSpeech* encargada de expresar por voz las cadenas de texto que se le envíen.

El componente *Actions* contiene la clase *Actions*, encargada de ejecutar las operaciones sobre el vehículo, así como mostrar por la pantalla y mediante voz las operaciones realizadas o los errores encontrados.

El componente *Bluetooth* contiene la clase *BluetoothConnection* encargada de realizar la conexión *Bluetooth* con el interfaz remoto conectado al vehículo y de mandar los datos que se le pida.

Como consideración final, se ha utilizado el patrón de diseño *Singleton* en las clases *BluetoothConnection*, *Actions* y *TextToSpeech*.

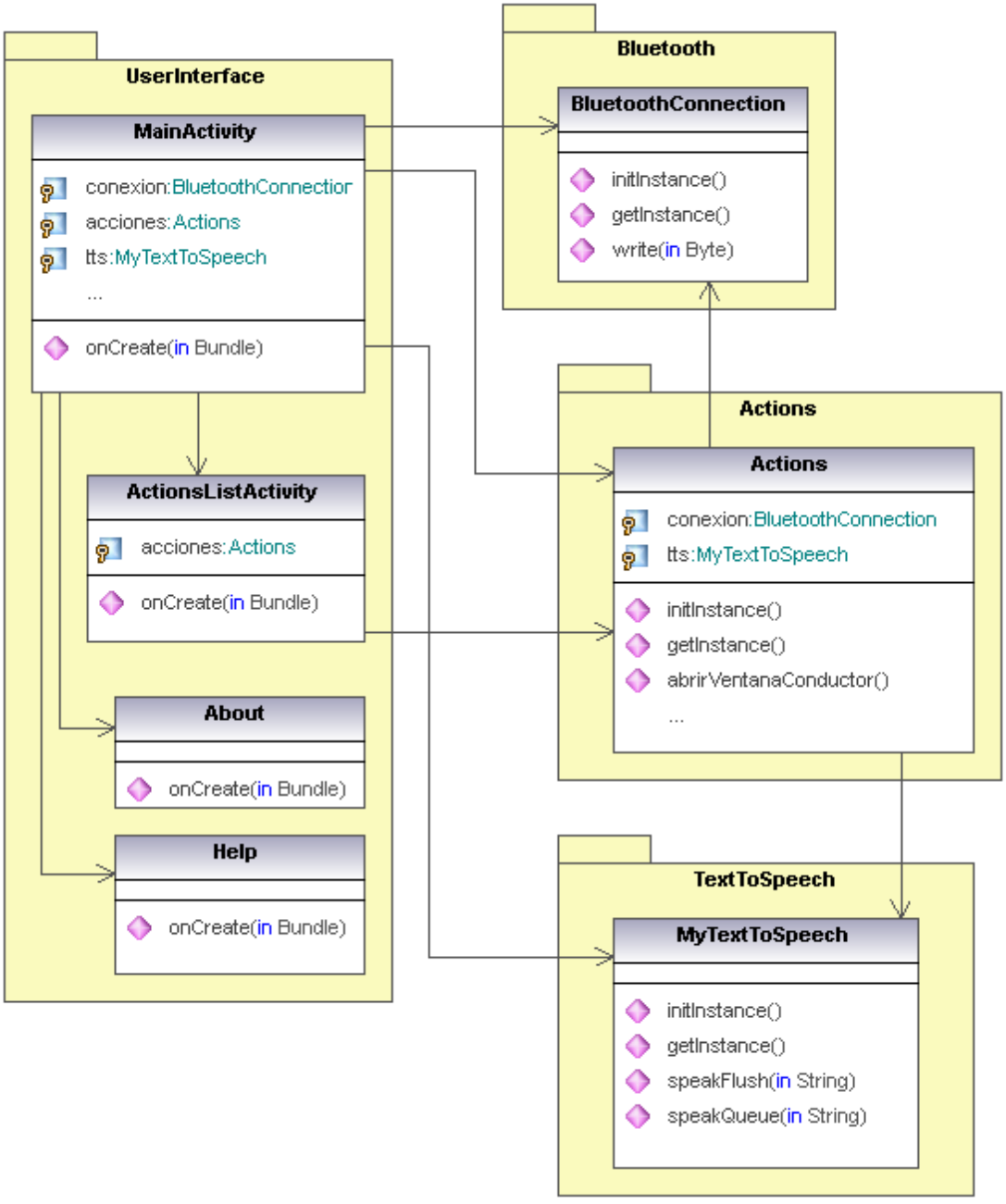


Figura 6.2: Diagrama de clases.

6.2.3. Interfaces de usuario

En este apartado se visualizan las distintas interfaces de usuario diseñadas para la aplicación y se explica cómo el usuario puede interactuar con ellas.

La aplicación cuenta con un icono de lanzamiento con el logotipo de la compañía BMW. Al pulsarse dicho icono se inicia la aplicación y muestra el primer interfaz. Si el dispositivo Bluetooth no está inicializado, la aplicación lo inicializa mientras muestra un cuadro de diálogo de progreso. La figura 6.3 muestra el icono de lanzamiento y el inicio de la aplicación.

Figura 6.3: Icono de lanzamiento e inicio de la aplicación.

Una vez que el dispositivo Bluetooth está activado, la aplicación realiza la conexión con el interfaz remoto instalado en el vehículo y permite al usuario utilizar el interfaz principal. La figura 6.4 muestra éste interfaz tanto con el móvil en posición horizontal como en vertical.

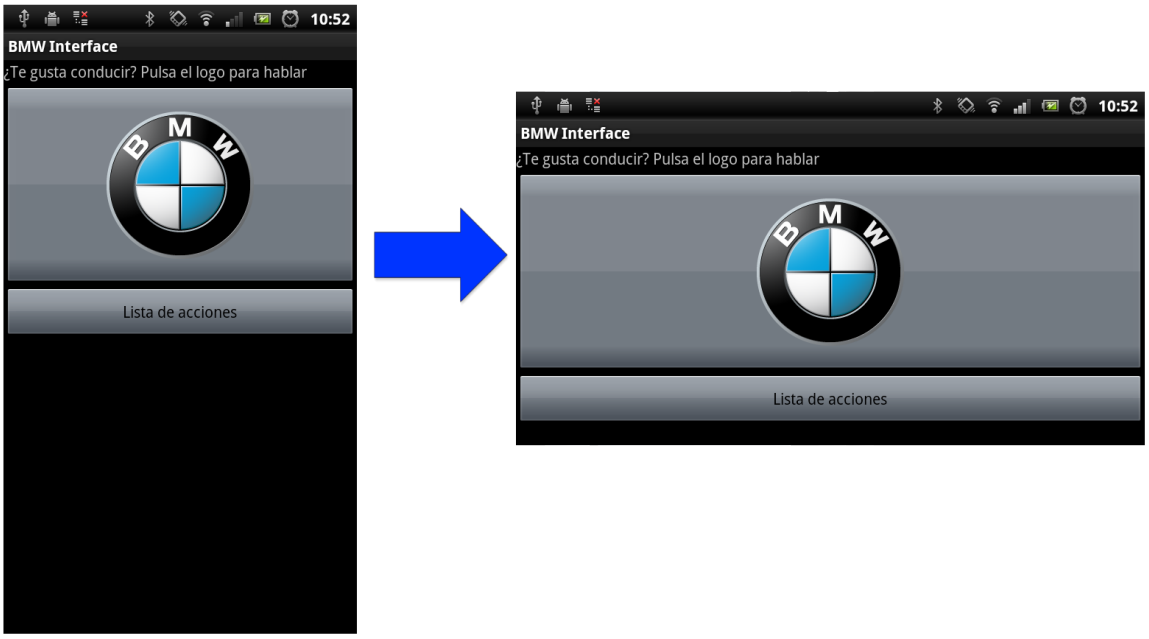


Figura 6.4: Interfaz principal de la aplicación en posición horizontal y vertical.

Desde el interfaz principal, se puede activar el reconocimiento de voz pulsando sobre el icono que muestra el logotipo de BMW, tal y como muestra la figura 6.5

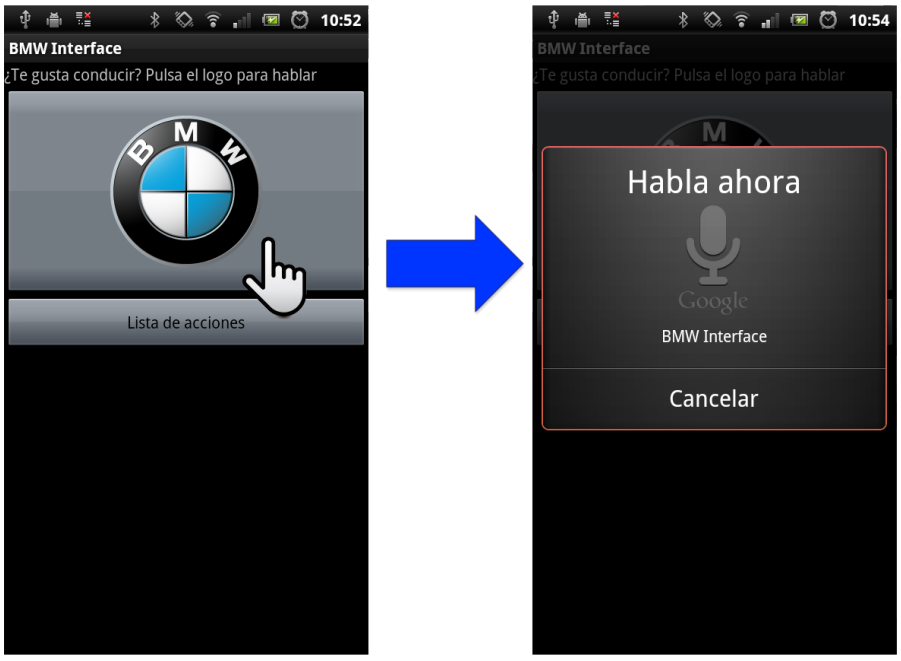


Figura 6.5: Reconocimiento de voz activado.

El interfaz principal también permite lanzar un segundo interfaz con un listado de todas las acciones que se pueden realizar sobre el vehículo. Cada elemento del listado en éste segundo interfaz puede pulsarse, realizándose la acción correspondiente a continuación. La figura 6.6 muestra la navegación hasta el segundo interfaz.

Figura 6.6: Accediendo al listado de acciones.

Por último, desde el interfaz principal también se puede acceder a un menú de opciones, pulsando para ello el botón de menú del móvil. Las opciones de éste menú son: información sobre el autor, ayuda y salir de la aplicación. La figura 6.7 muestra la navegación hasta el interfaz con información sobre el autor, mientras que la figura 6.8 muestra la navegación hasta el interfaz con la ayuda de la aplicación.

Figura 6.7: Accediendo a la información sobre el autor.

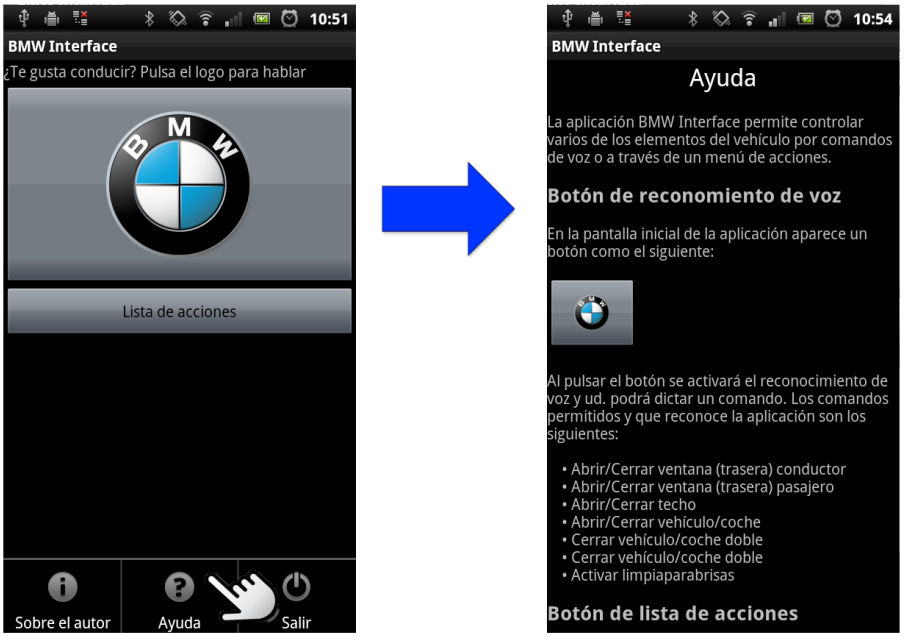


Figura 6.8: Accediendo a la ayuda de la aplicación.

6.3. Implementación

En esta sección se muestran los aspectos más relevantes de la implementación de la aplicación así como la instalación en el dispositivo móvil.

6.3.1. Fichero manifest

La aplicación necesita para su ejecución permisos para utilizar tanto la conexión de datos como el Bluetooth del terminal. En el fichero manifest es donde hay que declarar éstas intenciones, tal y como aparece en la figura 6.9.

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.INTERNET"/>
```

Figura 6.9: Fragmento del fichero *manifest* de la aplicación.

6.3.2. Interfaces de usuario (*layouts*)

En Android las interfaces de usuario se implementan cada una por separado en ficheros XML denominados *layouts*. El *layout* principal de la aplicación (*main.xml*) tiene la particularidad de no bloquear la pantalla del terminal, así como de mantener la imagen en todo momento en la pantalla. Éste layout cuenta con varios elementos a mostrar: el botón con la imagen de logo de BMW (para el reconocimiento de voz), el botón del listado de acciones, y distintos textos a lo largo de la pantalla.

La figura 6.10 muestra el comienzo del fichero main.xml con el layout principal. En dicha figura se pueden observar los atributos del layout, el texto que aparece sobre el botón principal, y el mismo botón principal con el logo de BMW.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:keepScreenOn="true"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/textoCabecera"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:paddingBottom="4dip"
        android:text="@string/textoCabecera" />

    <ImageButton
        android:id="@+id/botonLogo"
        android:layout_width="match_parent"
        android:layout_height="182dp"
        android:contentDescription="@string/botonReconocimiento"
        android:onClick="speakButtonClicked"
        android:src="@drawable/bmw_2000" />
```

Figura 6.10: Fragmento del *layout* principal de la aplicación.

6.3.3. Reconocimiento de voz

Para el reconocimiento de voz se han utilizado las funciones que tiene el *API* de Google para éste propósito. El reconocimiento se activa al pulsar el botón con el logo de BMW que aparece en el interfaz principal.

Internamente, el botón lanza un *intent* asociado a la clase del *API* de Google para el reconocimiento de voz, y a la acción de activar dicho reconocimiento (clase *RecognizerIntent*, acción *ACTION_RECOGNIZE_SPEECH*). Entre las opciones que se la pasan al *intent*, se indica que el reconocimiento sólo devuelva un resultado, que es el que se pasará a la función correspondiente de ejecución de comandos. La figura 6.11 muestra la creación del *intent*.

```
private void startVoiceRecognitionActivity()
{

    Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
            RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
    intent.putExtra(RecognizerIntent.EXTRA_PROMPT, "BMW Interface");
    intent.putExtra(RecognizerIntent.EXTRA_MAX_RESULTS, 1);
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, "ES");

    startActivityForResult(intent, REQUEST_CODE);

}
```

Figura 6.11: Creación del *intent* para el reconocimiento de voz.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    if (requestCode == REQUEST_CODE && resultCode == RESULT_OK)
    {
        String accion = null;

        ArrayList<String> matches = data.getStringArrayListExtra(
                RecognizerIntent.EXTRA_RESULTS);
        accion = matches.get(0);

        realizarAccion(accion);
    }
    super.onActivityResult(requestCode, resultCode, data);
}
```

Figura 6.12: Gestión del resultado del *activity* de reconocimiento de voz.

Una vez creado el *intent*, se ha de ejecutar una *activity* con dicho *intent* con la particularidad de devolver un resultado. Éste resultado es lo que el reconocimiento ha obtenido de la locución del usuario. La figura 6.12 muestra la función que gestiona el resultado devuelto por la ejecución del *activity* de reconocimiento de voz. Ésta función introduce el resultado en una cadena de texto y se lo pasa a la función de ejecución de comandos.

6.3.4. Síntesis de voz

Al igual que para el reconocimiento de voz, para la síntesis de voz se han utilizado las funciones que tiene el *API* de Google para éste propósito. La síntesis se produce cada vez que el usuario ejecuta algún comando, informando de que se va a llevar a cabo y también si se produce algún error.

La aplicación tiene una clase dedicada a la síntesis de voz, con un patrón *Singleton*, por lo que sólo se genera una instancia de dicha clase. De ésta forma, la instancia única inicializa la función de síntesis de voz una sola vez para que el resto de clases la utilicen.

Para inicializar la síntesis de voz, se crea un *intent* asociado a la clase del *API* de Google para la síntesis de voz, y a la acción de comprobar si están instalados los datos necesarios (clase *TextToSpeech*, acción *ACTION_CHECK_TTS_DATA*). Después de ejecutar la acción del *intent*, se necesita una instancia de la clase *TextToSpeech.OnInitListener* para notificar que se ha completado la inicialización.

En el caso de que los datos para la síntesis de voz no estén instalados, se crea otro *intent* asociado a la misma clase pero para instalarlos, y se lanza con un *activity* (acción *ACTION_INSTALL_TTS_DATA*).

La figura 6.13 muestra el constructor de la clase *MyTextToSpeech*, donde se inicializa la síntesis de voz.

```
//Constructor privado
private MyTextToSpeech(final Context context){

    /*Inicializa la síntesis de voz con un intent.
     * En caso de que no estén instalados los datos
     * necesarios, lanza un intent para realizar
     * la instalación
     */
    Intent checkTTSIntent = new Intent();

    checkTTSIntent.setAction(TextToSpeech.Engine.ACTION_CHECK_TTS_DATA);

    tts = new TextToSpeech(context, new TextToSpeech.OnInitListener() {

        public void onInit(int status) {
            // TODO Auto-generated method stub
            if (status == TextToSpeech.SUCCESS)
            {
                Locale locale = Locale.getDefault();

                // Comprueba si el lenguaje está disponible
                switch (tts.isLanguageAvailable(locale))
                {
                    case TextToSpeech.LANG_COUNTRY_VAR_AVAILABLE:

                        tts.setLanguage(locale);
                        break;
                    case TextToSpeech.LANG_MISSING_DATA:

                            Intent installIntent = new Intent();
                            installIntent.setAction(TextToSpeech.Engine.ACTION_INSTALL_TTS_DATA);
                            context.startActivity(installIntent);
                        break;

                }
            }
        }
    });
}
```

Figura 6.13: Constructor de la clase *MyTextToSpeech*.

Por último, para utilizar la síntesis de voz, se han creado dos funciones en la clase *MyTextToSpeech*: *speakFlush* y *speakQueue*. La primera de ellas dicta inmediatamente la cadena de texto que recibe, eliminando las que estuviesen en cola o procesándose en ese momento; la segunda función añade a la cola de cadenas a procesar la cadena recibida. La figura 6.14 muestra estas dos funciones de la clase *MyTextToSpeech*.

```
public void speakFlush(String texto){
    tts.speak(texto,TextToSpeech.QUEUE_FLUSH, null);
}

public void speakQueue(String texto){
    tts.speak(texto,TextToSpeech.QUEUE_ADD, null);
}
```

Figura 6.14: Funciones para la síntesis de voz de la clase *MyTextToSpeech*.

6.3.5. Acciones

Las acciones sobre el vehículo se realizan enviando una serie de códigos hexadecimales a través del Bluetooth del móvil hacia el interfaz remoto conectado al vehículo. La clase *Actions* que utiliza un patrón Singleton y por tanto tiene una sola instancia, es la encargada de enviar éstos códigos. Para ello cuenta con una instancia de la clase encargada de las comunicaciones Bluetooth y de una serie de funciones y variables que envían el correspondiente código hexadecimal según la acción a ejecutar sobre el vehículo.

La figura 6.15 muestra las variables internas de la clase con los códigos hexadecimales de las posibles operaciones sobre el vehículo.

```
/* Códigos hexadecimales de las acciones
 * sobre la centralita ZKE
 */
private final String ABRIR_VENTANA_CONDUCTOR = "55551103f040a2";
private final String CERRAR_VENTANA_CONDUCTOR = "55551103f0a84a";
private final String ABRIR_VENTANA_PASAJERO = "55551103f004e6";
private final String CERRAR_VENTANA_PASAJERO = "55551103f012f0";
private final String ABRIR_VENTANA_TRASERA_CONDUCTOR = "55551103f020c2";
private final String CERRAR_VENTANA_TRASERA_CONDUCTOR = "55551103f0c82a";
private final String ABRIR_VENTANA_TRASERA_PASAJERO = "55551103f010f2";
private final String CERRAR_VENTANA_TRASERA_PASAJERO = "55551103f006e4";
private final String ABRIR_TECHO ="55551103f0e002";
private final String CERRAR_TECHO ="55551103f008ea";
private final String ABRIR_VEHICULO ="55551103f202e2";
private final String CERRAR_VEHICULO ="55551103f201e1";
private final String CERRAR_VEHICULO_DOBLE_CIERRE ="55551102a3b0";
private final String ACTIVAR_LIMPIAPARABRISAS = "55551103f121c2";
```

Figura 6.15: Variables con los códigos hexadecimales de las operaciones sobre el vehículo.

Por otro lado, la figura 6.16 muestra el código de la función encargada de abrir la ventana del conductor. Ésta función envía el código hexadecimal almacenado como cadena de texto de la variable global ABRIR_VENTANA_CONDUCTOR a la clase encargada de las comunicaciones Bluetooth e informa al usuario de que va a realizar la operación mediante síntesis de voz, además de los errores que puedan producirse.

```
void abrirVentanaConductor(Context context){
    String texto = "Abriendo la ventana del conductor";

    toast(texto,context);
    tts.speakFlush(texto);


    if (!conexion.write(hex2Byte(ABRIR_VENTANA_CONDUCTOR)));
        errorConexion(context);
}
```

Figura 6.16: Función para bajar la ventana del conductor.

6.3.6. Bluetooth

La comunicación Bluetooth de la aplicación es gestionada por una instancia de la clase *BluetoothConnection* (la instancia es única dado que la clase *BluetoothConnection* cumple el patrón *Singleton*).

El dispositivo Bluetooth va a utilizarse como si hubiese un "cable virtual" entre el móvil y el módem Bluetooth del interfaz conectado al vehículo, por tanto se utilizará el perfil de comunicaciones de Bluetooth en serie (*Serial Port Profile*).

La gestión del dispositivo Bluetooth tiene varias clases del API de Android involucradas: por un lado la clase *BluetoothAdapter*, que representa el dispositivo físico de Bluetooth del móvil; por otro lado la clase *BluetoothDevice*, que representa el dispositivo físico Bluetooth remoto; y por último la clase *BluetoothSocket* que sirve para establecer un socket de comunicación con el dispositivo Bluetooth remoto. En el código de la clase *BluetoothConnection* se puede observar cómo sus variables globales utilizan las clases anteriores, tal y como aparece en el figura 6.17.

```
private static BluetoothSocket mmSocket;
private static BluetoothDevice mmDevice;
private static BluetoothAdapter mBluetoothAdapter;
private OutputStream mmOutStream;
```

Figura 6.17: Variables globales de la clase *BluetoothConnection*.

Los pasos que realiza la clase *BluetoothConnection* para establecer una conexión Bluetooth son los siguientes:

- **Inicialización**: el dispositivo local Bluetooth ha de ser inicializado en caso de no estarlo previamente. Para ello se crea una instancia de *BluetoothAdapter*, y se comprueba si el dispositivo local está o no inicializado. En caso negativo se inicializa, y además se habilita el modo *discovery*. El código que realiza estas acciones puede verse en la figura 6.18.

```
mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

if (!mBluetoothAdapter.isEnabled()) {

    mBluetoothAdapter.enable();
    mBluetoothAdapter.startDiscovery();

}
```

Figura 6.18: Inicialización del dispositivo Bluetooth local.

- **Vincular con el dispositivo remoto**: el siguiente paso es establecer el vínculo (*pairing* en inglés) con el módem Bluetooth del interfaz remoto del vehículo, usando para ello su MAC. Mediante el código de la figura 6.19 se establece dicho vínculo, preguntando al usuario por el PIN del dispositivo remoto en caso de ser la primera vez que se establece. Como resultado del vínculo, se obtiene una instancia de la clase *BluetootDevice*.

```
mmDevice =  mBluetoothAdapter.getRemoteDevice("00:06:66:43:45:21");
```

Figura 6.19: Establecimiento del vínculo con el dispositivo Bluetooth remoto.

- **Abrir un *socket***: una vez realizado el vínculo con el dispositivo remoto, se ha de abrir un *socket* de conexión para permitir el flujo de datos entre éste último y el dispositivo local de Bluetooth. Para ello, se ha de utilizar el método *createRfcommSocketToServiceRecord* con la instancia de *BluetootDevice*. Como resultado de su ejecución, el método abre un *socket* seguro de conexión y lo devuelve a través de una instancia de la clase *BluetoothSocket*, según se puede observar en el código de la figura 6.20.

```
// UUID de la aplicación
MY_UUID = "00001101-0000-1000-8000-00805F9B34FB";

// Abre un socket seguro contra el modem bluetooth del interfaz remoto
try {
    mmSocket = mmDevice.createRfcommSocketToServiceRecord
            (UUID.fromString(MY_UUID));

} catch (IOException e) {}
```

Figura 6.20: Creación del socket con el dispositivo Bluetooth remoto.

- **Conectar el *socket***: por último, para establecer la conexión Bluetooth hay que conectar el *socket* creado en el paso anterior. Para ello se utiliza la función *connect* sobre la instancia de la clase *BluetoothSocket*. Después hay

que obtener del *socket* un *stream* de salida, y asociarlo a una instancia de *OutputStream*, de ésta forma se podrán enviar datos a través del *socket*. La figura 6.21 muestra el código de éstas operaciones.

```
try {

    // Conecta el dispositivo a través del socket.
    // Se bloquea hasta conseguir el socket o lanzar una excepción
    mmSocket.connect();
    socketConnected = true;

    mmOutStream = mmSocket.getOutputStream();

} catch (Exception connectException) {
    Log.e(LOGTAG, "Fallo en la conexión con el socket: "+connectException);
    // No se ha podido conectar, se cierra el socket
    try {
        mmSocket.close();
    } catch (Exception closeException) {}
    socketConnected = false;
}
```

Figura 6.21: Conexión del socket Bluetooth.

Una vez creada la conexión, los datos se envían a través de la instancia de *OutputStream* (variable *mmOutStream*), tal y como se puede observar en el fragmento de código de la figura 6.22.

```
try {
    mmOutStream.write(bytes);
    exito = true;
} catch (Exception e) {
     Log.e(LOGTAG, "Error en el envio: "+e);
}
```

Figura 6.22: Escritura a través del socket Bluetooth.

6.3.7. Instalación en el dispositivo móvil

La instalación de la aplicación en el teléfono móvil se hace desde el propio Eclipse configurado con el plugin ADT de Android. Al ejecutar desde Eclipse la aplicación, permite seleccionar si se quiere ejecutar directamente en el móvil, o bien en un simulador de Android.

Si se opta por la ejecución en el móvil, se ha de configurar en él la conexión USB para que utilice el modo transferencia multimedia (MTP). Después se conecta al ordenador donde esté instalado el Eclipse y tenga configurados los drivers para dicho móvil.

En Eclipse, hay que abrir el proyecto donde esté implementada la aplicación, como muestra la figura 6.23.

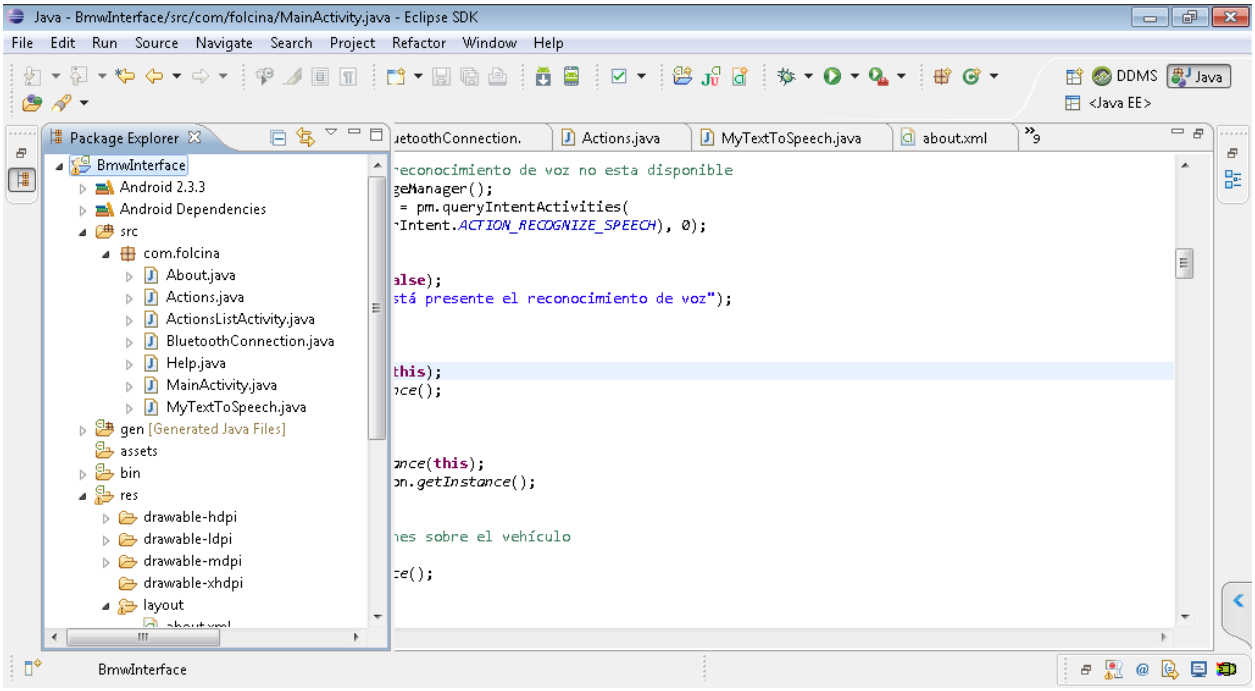


Figura 6.23: Desarrollo de la aplicación BmwInterface en Eclipse.

Una vez abierto el proyecto, se pulsa sobre el menú superior en *Run* y después en *Run Configurations...*, como se puede observar en la figura 6.24.

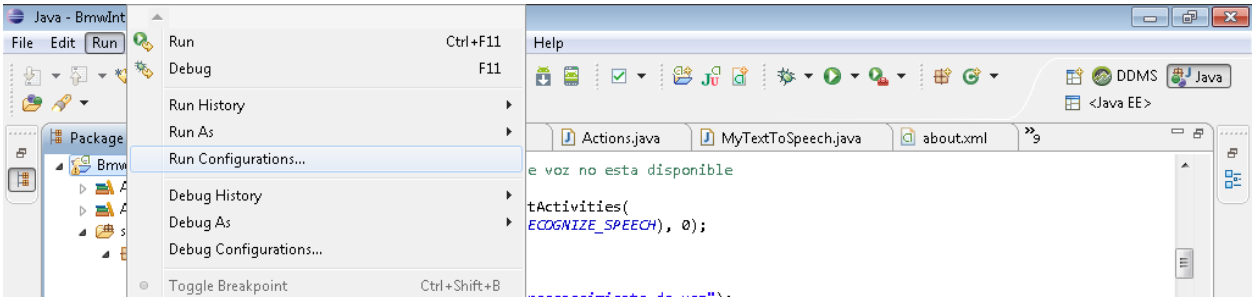


Figura 6.24: Menú de ejecución de Eclipse.

Después de acceder a dicho menú, se abrirá una ventana con la configuración de ejecución, como aparece en la figura 6.25. En dicha ventana se pulsará sobre la pestaña *target* y se seleccionará *manual* en el apartado de *Deployment Target Selection Mode*. Por último se pulsa sobre *Apply* y después sobre *Run*.

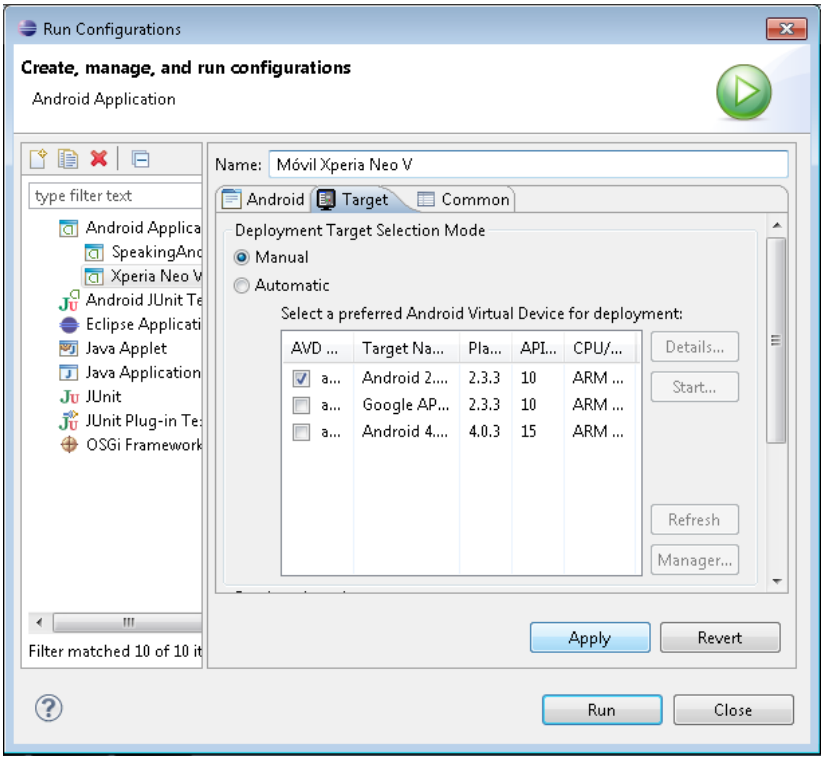


Figura 6.25: Ventana *Run Configurations* de Eclipse.

La siguiente ventana muestra el terminal móvil y su número de serie, además de si está o no operativo (*online*). Para ejecutar la aplicación sobre él, se debe seleccionar y pulsar sobre *OK*, como se muestra en la figura 6.26. A partir de ese momento, la aplicación se instala en el móvil permanentemente y pasa a ejecutarse.

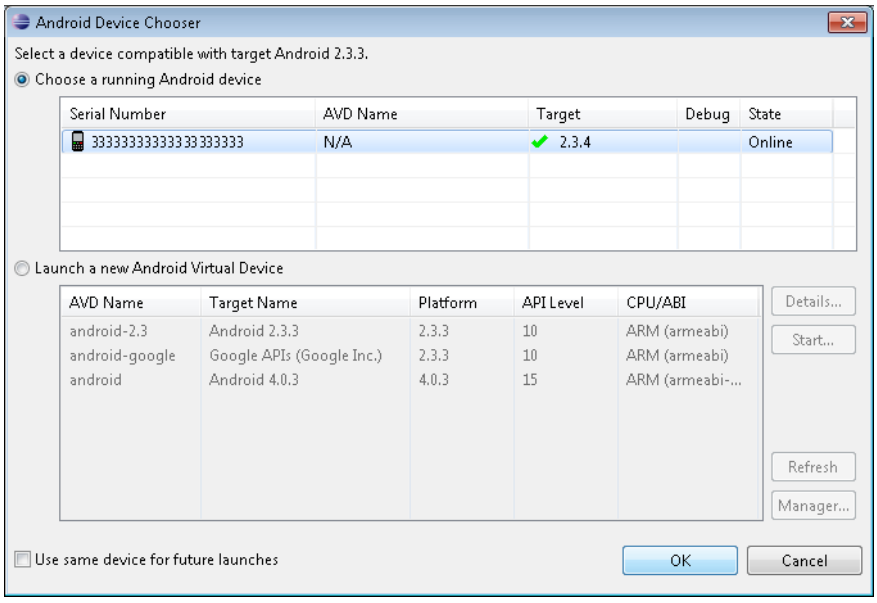


Figura 6.26: Ventana de selección del dispositivo Android donde ejecutar la aplicación.

Capítulo 7

Integración y pruebas

En éste capítulo se realiza la integración del sistema con el vehículo y se procede a realizar las pruebas correspondientes para verificar que el sistema cumple todos los requisitos iniciales.

7.1. Integración

La integración del sistema consiste en conectar el interfaz de conexión remota al conector de diagnosis del vehículo. Ésta integración no es definitiva, dado que la conexión se hace para comprobar el correcto funcionamiento del sistema sin dejar las diferentes partes conectadas de forma permanente.

Para la conexión permanente se requiere fijar tanto el interfaz como el cable que lo conecta al vehículo dentro del vano motor. Ésta tarea será realizada como trabajo futuro.

La figura 7.1 muestra el interfaz de conexión remota integrado en el vehículo para la realización de las pruebas. El interfaz esta señalado por una flecha roja y aparece abierto para que se observe el PCB en su interior. Está unido al vehículo a través de un cable que se conecta al propio interfaz (mediante un conector serie DB9) y al vehículo por su toma de diagnosis indicada mediante una flecha verde.

Una vez integrado, el interfaz de conexión remota recibe la alimentación a través del conector de diagnosis. Mientras permanezca conectado estará activado y a la espera de solicitudes de conexión.

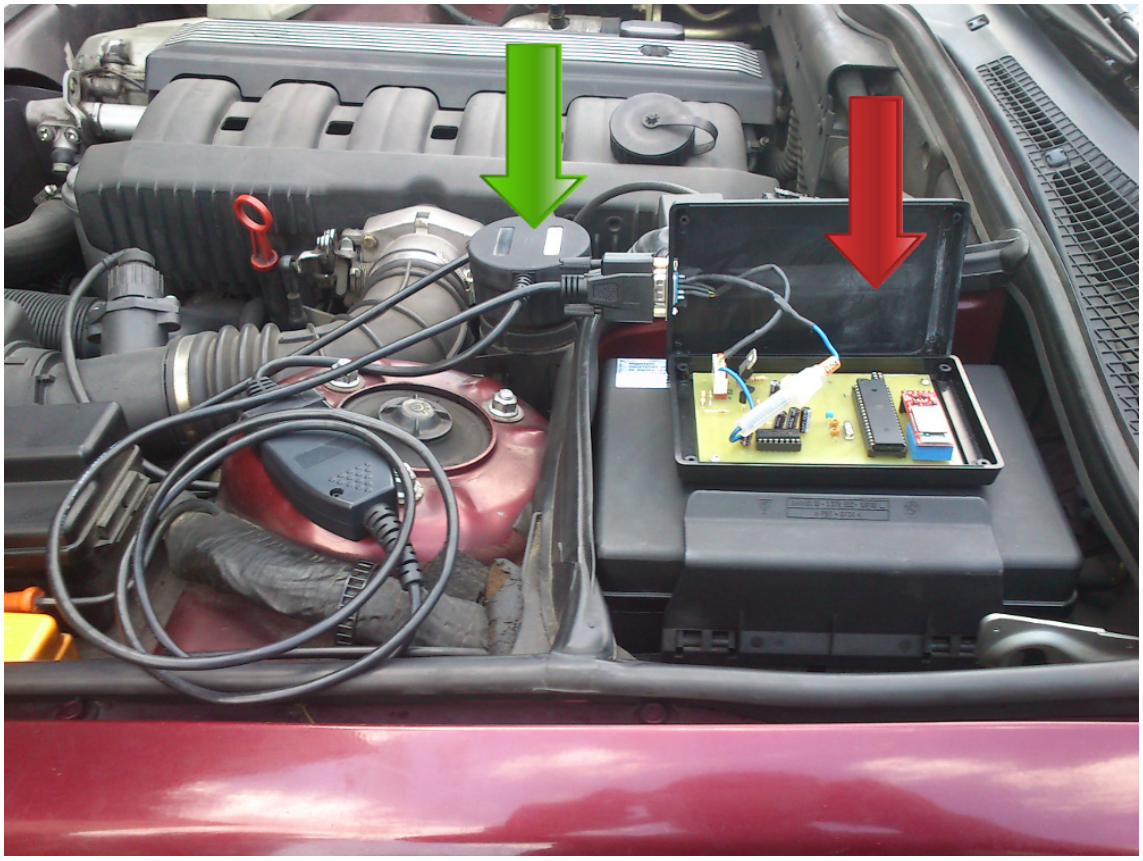


Figura 7.1: Integración del interfaz de conexión remota en el vehículo.

7.2. Pruebas

Las pruebas se dividen en pruebas de integración, donde se verifican aspectos sobre el comportamiento del sistema en la integración, y pruebas de aceptación, donde se verifica que se cumplan todos los requisitos de capacidad de la fase de análisis.

La información que contiene cada prueba es la siguiente:

- **Identificador**: se trata de un identificador único para cada prueba. Las pruebas de integración tendrán un identificador con la forma PRU-INXX, mientras que las pruebas de aceptación tendrán un identificador con la forma PRU-ACXX. En ambos casos, XX representa un número decimal que sirve para enumerar de forma secuencial las pruebas.
- **Nombre**: define un nombre para la prueba.
- **Descripción**: se realiza una descripción detallada de la prueba.
- **Procedimiento**: se detallan los pasos para la realización de la prueba.
- **Resultado** (sólo para las pruebas de integración): expone los resultados

obtenidos en la prueba.

- **Requisito de capacidad relacionado** (sólo para las pruebas de aceptación): identifica el requisito de capacidad que da origen a la prueba de aceptación.
- **Resultado esperado** (sólo para las pruebas de aceptación): expone el resultado que se espera en la realización de la prueba.
- **Estado**: indica si se ha realizado o no la prueba con éxito.

7.2.1. Pruebas de integración

A continuación se listan las pruebas de integración:

Identificador	PRU-IN01
Nombre	Tensión del circuito
Descripción	Se comprueba la tensión existente en el interfaz de conexión remota cuando está conectado al vehículo
Procedimiento	▪ Se utiliza un multímetro y se coloca en la posición de medir tensión en corriente continua en la escala de 20V. ▪ Se coloca la punta de tierra del multímetro (de color negro) sobre la patilla de tierra del regulador de tensión LM7805. ▪ Se coloca la punta de tensión del multímetro (de color rojo) sobre la patilla de entrada del regulador de tensión LM7805. ▪ Se observa el valor que da el multímetro con las puntas colocadas en los puntos anteriores.
Resultado	La tensión que llega al interfaz es de +12,2V.
Estado	Prueba realizada con éxito.

Cuadro 7.1: Prueba de integración número 1.

Identificador	PRU-IN02
Nombre	Intensidad del circuito.
Descripción	Se comprueba la intensidad existente en el interfaz de conexión remota cuando está conectado al vehículo.
Procedimiento	▪ Se utiliza un multímetro y se pone en la posición de medir intensidad de corriente continua, en la escala de 10 amperios. El cable rojo se conecta en el multímetro en el conector de medir intensidad de corriente hasta 10 amperios. ▪ Se desconecta el interfaz del vehículo, después se retira el fusible y se utilizan los extremos del portafusibles para poner el multímetro en serie con el circuito, según muestra la figura 7.2. ▪ Se conecta de nuevo el interfaz al vehículo y se realizan varias mediciones en los estados de sin conexión al móvil, conectado a la aplicación del móvil y realizando una acción sobre el vehículo.
Resultado	La intensidad aproximada del interfaz de conexión remota es de: 20 mA mínima, 60 mA máxima.
Estado	Prueba realizada con éxito.

Cuadro 7.2: Prueba de integración número 2.

Figura 7.2: Medición de la intensidad de corriente en el interfaz de conexión remota.

7.2.2. Pruebas de aceptación

A continuación se listan las pruebas de aceptación:

Identificador	PRU-AC01
Nombre	Abrir ventanillas
Descripción	Se comprueba que el usuario puede abrir las ventanillas del vehículo desde la aplicación, seleccionando la acción a través de un listado o mediante comandos de voz.
Procedimiento	▪ El usuario inicia la aplicación. ▪ El usuario abre el menú de acciones. ▪ El usuario pulsa sobre "Abrir la ventana del conductor". ▪ El usuario pulsa sobre "Abrir la ventana del pasajero". ▪ El usuario pulsa sobre "Abrir la ventana trasera del conductor". ▪ El usuario pulsa sobre "Abrir la ventana trasera del pasajero". ▪ El usuario vuelve a la pantalla principal. ▪ El usuario pulsa el icono de reconocimiento de voz y dicta la orden: "Abrir la ventana del conductor". ▪ El usuario pulsa el icono de reconocimiento de voz y dicta la orden: "Abrir la ventana del pasajero". ▪ El usuario pulsa el icono de reconocimiento de voz y dicta la orden: "Abrir la ventana trasera del conductor". ▪ El usuario pulsa el icono de reconocimiento de voz y dicta la orden: "Abrir la ventana trasera del pasajero".
Requisito de capacidad relacionado	RUC-01
Resultado esperado	Cada vez que el usuario pulsa sobre una de las acciones de abrir la ventanilla u ordena abrir dicha ventanilla con un comando de voz, ha de abrirse en el vehículo la ventanilla correspondiente.
Estado	Prueba realizada con éxito.

Cuadro 7.3: Prueba de aceptación número 1.

Identificador	PRU-AC02
Nombre	Cerrar ventanillas
Descripción	Se comprueba que el usuario puede cerrar las ventanillas del vehículo desde la aplicación, seleccionando la acción a través de un listado o mediante comandos de voz.
Procedimiento	▪ El usuario inicia la aplicación. ▪ El usuario abre el menú de acciones. ▪ El usuario pulsa sobre "Cerrar la ventana del conductor". ▪ El usuario pulsa sobre "Cerrar la ventana del pasajero". ▪ El usuario pulsa sobre "Cerrar la ventana trasera del conductor". ▪ El usuario pulsa sobre "Cerrar la ventana trasera del pasajero". ▪ El usuario vuelve a la pantalla principal. ▪ El usuario pulsa el icono de reconocimiento de voz y dicta la orden: "Cerrar la ventana del conductor". ▪ El usuario pulsa el icono de reconocimiento de voz y dicta la orden: "Cerrar la ventana del pasajero". ▪ El usuario pulsa el icono de reconocimiento de voz y dicta la orden: "Cerrar la ventana trasera del conductor". ▪ El usuario pulsa el icono de reconocimiento de voz y dicta la orden: "Cerrar la ventana trasera del pasajero".
Requisito de capacidad relacionado	RUC-02
Resultado esperado	Cada vez que el usuario pulsa sobre una de las acciones de cerrar la ventanilla u ordena cerrar dicha ventanilla con un comando de voz, ha de cerrarse en el vehículo la ventanilla correspondiente.
Estado	Prueba realizada con éxito.

Cuadro 7.4: Prueba de aceptación número 2.

Identificador	PRU-AC03
Nombre	Abrir techo.
Descripción	Se comprueba que el usuario puede abrir el techo eléctrico del vehículo desde la aplicación, seleccionando la acción a través de un listado o mediante comandos de voz.
Procedimiento	▪ El usuario inicia la aplicación. ▪ El usuario abre el menú de acciones. ▪ El usuario pulsa sobre "Abrir techo". ▪ El usuario vuelve a la pantalla principal. ▪ El usuario pulsa el icono de reconocimiento de voz y dicta la orden: "Abrir techo".
Requisito de capacidad relacionado	RUC-03
Resultado esperado	Cada vez que el usuario pulsa sobre la acción de abrir el techo u ordena abrir el techo con un comando de voz, ha de abrirse el techo eléctrico del vehículo.
Estado	Prueba realizada con éxito.

Cuadro 7.5: Prueba de aceptación número 3.

Identificador	PRU-AC04
Nombre	Cerrar techo
Descripción	Se comprueba que el usuario puede cerrar el techo eléctrico del vehículo desde la aplicación, seleccionando la acción a través de un listado o mediante comandos de voz.
Procedimiento	▪ El usuario inicia la aplicación. ▪ El usuario abre el menú de acciones. ▪ El usuario pulsa sobre "Cerrar techo". ▪ El usuario vuelve a la pantalla principal. ▪ El usuario pulsa el icono de reconocimiento de voz y dicta la orden: "Cerrar techo".
Requisito de capacidad relacionado	RUC-04
Resultado esperado	Cada vez que el usuario pulsa sobre la acción de cerrar el techo u ordena cerrar el techo con un comando de voz, ha de cerrarse el techo eléctrico del vehículo.
Estado	Prueba realizada con éxito.

Cuadro 7.6: Prueba de aceptación número 4.

Identificador	PRU-AC05
Nombre	Abrir vehículo
Descripción	Se comprueba que el usuario puede abrir el vehículo desde la aplicación, seleccionando la acción a través de un listado o mediante comandos de voz.
Procedimiento	▪ El usuario inicia la aplicación. ▪ El usuario abre el menú de acciones. ▪ El usuario pulsa sobre "Abrir vehículo". ▪ El usuario vuelve a la pantalla principal. ▪ El usuario pulsa el icono de reconocimiento de voz y dicta la orden: "Abrir vehículo".
Requisito de capacidad relacionado	RUC-05
Resultado esperado	Cada vez que el usuario pulsa sobre la acción de abrir el vehículo u ordena abrir el vehículo con un comando de voz, ha de realizarse la operación de apertura del vehículo.
Estado	Prueba realizada con éxito.

Cuadro 7.7: Prueba de aceptación número 5.

Identificador	PRU-AC06
Nombre	Cerrar vehículo
Descripción	Se comprueba que el usuario puede cerrar el vehículo desde la aplicación, seleccionando la acción a través de un listado o mediante comandos de voz.
Procedimiento	▪ El usuario inicia la aplicación. ▪ El usuario abre el menú de acciones. ▪ El usuario pulsa sobre "Cerrar vehículo". ▪ El usuario vuelve a la pantalla principal. ▪ El usuario pulsa el icono de reconocimiento de voz y dicta la orden: "Cerrar vehículo".
Requisito de capacidad relacionado	RUC-06
Resultado esperado	Cada vez que el usuario pulsa sobre la acción de cerrar el vehículo u ordena cerrar el vehículo con un comando de voz, ha de realizarse la operación de cierre del vehículo.
Estado	Prueba realizada con éxito.

Cuadro 7.8: Prueba de aceptación número 6.

Identificador	PRU-AC07
Nombre	Cierre seguro del vehículo
Descripción	Se comprueba que el usuario puede cerrar el vehículo con doble cierre desde la aplicación, seleccionando la acción a través de un listado o mediante comandos de voz.
Procedimiento	▪ El usuario inicia la aplicación. ▪ El usuario abre el menú de acciones. ▪ El usuario pulsa sobre "Cerrar vehículo con doble cierre". ▪ El usuario vuelve a la pantalla principal. ▪ El usuario pulsa el icono de reconocimiento de voz y dicta la orden: "Cerrar vehículo con doble cierre".
Requisito de capacidad relacionado	RUC-07
Resultado esperado	Cada vez que el usuario pulsa sobre la acción de cerrar el vehículo con doble cierre u ordena dicha acción con un comando de voz, ha de realizarse la operación de cierre del vehículo con doble cierre.
Estado	Prueba realizada con éxito.

Cuadro 7.9: Prueba de aceptación número 7.

Identificador	PRU-AC08
Nombre	Activar limpiaparabrisas
Descripción	Se comprueba que el usuario puede activar el limpiaparabrisas del vehículo desde la aplicación, seleccionando la acción a través de un listado o mediante comandos de voz.
Procedimiento	▪ El usuario inicia la aplicación. ▪ El usuario abre el menú de acciones. ▪ El usuario pulsa sobre "Activar el limpiaparabrisas'. ▪ El usuario vuelve a la pantalla principal. ▪ El usuario pulsa el icono de reconocimiento de voz y dicta la orden: "Activar el limpiaparabrisas".
Requisito de capacidad relacionado	RUC-08
Resultado esperado	Cada vez que el usuario pulsa sobre la acción de activar el limpiaparabrisas u ordena dicha acción con un comando de voz, ha de activarse el limpiaparabrisas del vehículo.
Estado	Prueba realizada con éxito.

Cuadro 7.10: Prueba de aceptación número 8.

Identificador	PRU-AC09
Nombre	Reconocimiento de voz
Descripción	Se comprueba que la aplicación sea capaz de reconocer comandos de voz y ejecutar sobre el vehículo las acciones reconocidas y que forman parte de la aplicación
Procedimiento	▪ El usuario inicia la aplicación. ▪ El usuario pulsa el icono de reconocimiento de voz y dicta una orden del listado de órdenes disponibles. ▪ El usuario vuelve a realizar el punto anterior hasta que haya dictado todas las órdenes disponibles.
Requisito de capacidad relacionado	RUC-09
Resultado esperado	Cada vez que el usuario dicta una orden, se ha de reconocer dicha orden y ejecutarla si está disponible en el sistema.
Estado	Prueba realizada con éxito.

Cuadro 7.11: Prueba de aceptación número 9.

Identificador	PRU-AC10
Nombre	Ejecución a través de pantalla táctil
Descripción	Se comprueba que el usuario es capaz de ejecutar las acciones sobre el vehículo pulsando sobre ellas en un listado de acciones.
Procedimiento	▪ El usuario inicia la aplicación. ▪ El usuario pulsa sobre el botón de "Lista de acciones". ▪ El usuario pulsa sobre cada una de las acciones que aparecen en el listado.
Requisito de capacidad relacionado	RUC-10
Resultado esperado	Cada vez que el usuario pulsa sobre una de las acciones ésta ha de realizarse sobre el vehículo.
Estado	Prueba realizada con éxito.

Cuadro 7.12: Prueba de aceptación número 10.

Identificador	PRU-AC11
Nombre	Mostrar texto al usuario
Descripción	Se comprueba que aparezca un texto informando de la acción a realizar cuando el usuario ejecute alguna acción.
Procedimiento	▪ El usuario inicia la aplicación. ▪ El usuario pulsa sobre el botón de "Lista de acciones". ▪ El usuario pulsa sobre cada una de las acciones que aparecen en el listado. ▪ El usuario regresa a la pantalla principal. ▪ El usuario pulsa el icono de reconocimiento de voz y dicta una orden del listado de órdenes disponibles. ▪ El usuario vuelve a realizar el punto anterior hasta que haya dictado todas las órdenes disponibles.
Requisito de capacidad relacionado	RUC-11
Resultado esperado	Cuando el usuario ejecuta una acción, ya sea por comandos de voz o pulsando sobre alguna acción del listado de acciones, la aplicación mostrará un texto informando de la operación a realizar.
Estado	Prueba realizada con éxito.

Cuadro 7.13: Prueba de aceptación número 11.

Identificador	PRU-AC12
Nombre	Expresión por voz
Descripción	Se comprueba que la aplicación indique mediante voz la acción que ejecute el usuario.
Procedimiento	▪ El usuario inicia la aplicación. ▪ El usuario pulsa sobre el botón de "Lista de acciones". ▪ El usuario pulsa sobre cada una de las acciones que aparecen en el listado. ▪ El usuario regresa a la pantalla principal. ▪ El usuario pulsa el icono de reconocimiento de voz y dicta una orden del listado de órdenes disponibles. ▪ El usuario vuelve a realizar el punto anterior hasta que haya dictado todas las órdenes disponibles.
Requisito de capacidad relacionado	RUC-12
Resultado esperado	Cuando el usuario ejecuta una acción, ya sea por comandos de voz o pulsando sobre alguna acción del listado de acciones, la aplicación informará mediante voz de la operación a realizar.
Estado	Prueba realizada con éxito.

Cuadro 7.14: Prueba de aceptación número 12.

Identificador	PRU-AC13
Nombre	Informar de error
Descripción	Se comprueba que la aplicación informe de cualquier error que se produzca al intentar ejecutar una acción.
Procedimiento	▪ El usuario inicia la aplicación. ▪ El usuario pulsa el icono de reconocimiento de voz y dicta una orden que no aparece en el listado de órdenes disponibles. ▪ Se desconecta el interfaz del vehículo. ▪ El usuario pulsa sobre el botón de "Lista de acciones". ▪ El usuario pulsa sobre cualquiera de las acciones disponibles en el listado.
Requisito de capacidad relacionado	RUC-13
Resultado esperado	Se espera que cuando el usuario dicte una orden que no está disponible, le informe de que dicha orden no es válida, incluyendo la propia orden del usuario. Además, cuando se desconecte el interfaz del vehículo, se espera que al intentar ejecutar una orden en el listado de ordenes, la aplicación informe de que hay un error con la conexión al vehículo.
Estado	Prueba realizada con éxito.

Cuadro 7.15: Prueba de aceptación número 13.

Capítulo 8

Conclusiones y trabajos futuros

En éste capítulo se exponen las conclusiones obtenidas a la finalización del presente proyecto y los trabajos futuros a realizar relacionados con el sistema desarrollado y el conocimiento adquirido para su realización.

8.1. Conclusiones

8.1.1. Conclusiones del producto

El proyecto ha consistido en el desarrollo de un sistema que permite controlar de forma remota varios elementos de la carrocería de un vehículo BMW. El sistema se divide en dos bloques: una parte hardware, consistente en un circuito impreso conectado al vehículo; y una parte software, consistente en una aplicación para móviles con S.O Android.

Los elementos del vehículo que son controlados por el sistema son: las ventanas, el cierre centralizado, el techo eléctrico y el limpiaparabrisas. Todos estos elementos pueden ser accionados desde la aplicación de manera manual o por comandos de voz.

El sistema se ha desarrollado en 6 meses aproximadamente y las pruebas realizadas confirman que cumple con éxito todos los requisitos iniciales.

Uno de los propósitos del sistema ha sido el de mejorar el confort de conducción, al poder manipular mediante comandos de voz los distintos elementos de la carrocería. No se conoce ningún sistema que venga instalado de serie en los vehículos actuales que permita realizar operaciones similares por voz, por lo tanto, el proyecto es innovador en sus objetivos.

Es destacable el hecho de haberse realizado sobre un vehículo fabricado en el año 1993 (contemporáneo a la introducción del procesador Intel Pentium), y que no haya sido necesario modificar la electrónica del vehículo. El sistema se ha conectado a una toma de diagnosis y gracias a las funcionalidades ya existentes

en las centralitas, se ha logrado controlar de forma remota los elementos de la carrocería mencionados anteriormente.

Es presumible que en los vehículos actuales las centralitas puedan tener las mismas funcionalidades permitiendo crear sistemas similares al del presente proyecto. Las compañías de automóviles deben tener algún interés especial en no haber incluido controles de voz para los cierres y ventanas de sus vehículos.

8.1.2. Conclusiones del proceso

Una vez establecidos los objetivos y requisitos del sistema (fase de análisis), se ha realizado un estudio de la electrónica del vehículo. Para ello se ha contado con abundante documentación que aparece en la bibliografía de ésta memoria.

Mediante dicho estudio se han tenido los conocimientos necesarios para poder realizar un estudio práctico de las transmisiones realizadas por dos sistemas completos de diagnosis: Carsoft y Ediabas ToolSet 32. En éste estudio práctico, se ha utilizado una centralita obtenida de un desguace igual a la del vehículo, haciéndola funcionar sobre una mesa de trabajo. Sobre dicha centralita se han utilizado los sistemas de diagnosis mencionados aplicado ingeniería inversa para conocer los códigos que se transmitían en la comunicación.

Una vez que se concluidos los estudios anteriores, se ha fabricado un circuito impreso de forma casera y completamente funcional. Se han utilizado materiales económicos y en algunos casos se han reutilizado componentes de aparatos eléctricos en desuso (por ejemplo el portafusibles aéreo se ha obtenido de una radio).

Por último se ha desarrollado una aplicación para móviles con S.O Android. Ésta aplicación hace uso del API de Android para utilizar el Bluetooth del móvil, el reconocimiento de voz y la síntesis de voz.

8.1.3. Conclusiones personales

El proyecto me ha servido para poner en práctica conocimientos ya adquiridos con anterioridad como lo aprendido durante la carrera o la elaboración de circuitos impresos, y también para aprender acerca de tecnologías que me interesaban como es el caso de Android o los sistemas de diagnosis de los vehículos BMW.

He utilizado mi propio vehículo para el proyecto con idea de conocerlo más en detalle, además de poder disfrutar personalmente del sistema desarrollado.

8.2. Trabajos futuros

El proyecto forma parte de otro proyecto más global y personal de mejora y restauración del vehículo utilizado. Se seguirá trabajando en el sistema desarrollado e integrándolo con nuevos sistemas. Las posibilidades que se contemplan son las siguientes:

- **Acerca del interfaz de conexión remoto**:
  - Se desea fijar de forma permanente el interfaz y el cable externo que lo conecta al vehículo de forma permanente, utilizando los soportes necesarios y guiando el cable para que no interfiera con los elementos que hay alojados en el vano del motor.
  - El interfaz tiene un papel pasivo en éste proyecto, se desea ampliar su funcionalidad y que realice gestiones de manera autónoma, como por ejemplo, comprobar el estado de las centralitas y avisar de algún fallo al dispositivo móvil, o bloquear las puertas después de alcanzar una determinada velocidad.
    Realizar éste cambio no requiere modificar el circuito, al tener un diseño escalable gracias a la existencia del microcontrolador; con la reprogramación de éste último se conseguirían las nuevas funcionalidades.
  - Incorporar al circuito un sensor de temperatura, para conocer la temperatura del vano del motor mientras se está utilizando el vehículo y avisar cuando se sobrepase la temperatura máxima de trabajo del circuito (los componentes funcionan de manera óptima hasta 70 grados centígrados).
- **Acerca de la aplicación**:
  - Seguir realizando estudios para encontrar los códigos que activen más elementos del vehículo e incluso reprogramar la información de las centralitas sobre ciertos aspectos, como resetear los intervalos de servicio, todo ello desde la aplicación.
  - Dado que la aplicación se centra en el control remoto de ciertos elementos, sería deseable ampliar la funcionalidad de la aplicación para poder hacer diagnosis de todas las centralitas del vehículo. Bastaría con incorporar nuevas pantallas a la aplicación que envíen los datos pertinentes al interfaz remoto.
  - Mostrar datos en tiempo real sobre aspectos del vehículo, por ejemplo, la velocidad a la que se circula, o las revoluciones del motor.
- **Integración con otros sistemas**:
  - Se desea construir en un futuro una nueva alarma que sustituya a la ya existente en el vehículo. Esta nueva alarma utilizaría el interfaz remoto para consultar datos de las centralitas o activar elementos. También

podría comunicarse con la aplicación de móvil, y desde la misma activar o desactivar la alarma mientras se realiza el cierre o apertura del vehículo.

- También se desea construir un segundo ordenador de abordo [1] que tenga una pantalla táctil y sistema operativo Android. El ordenador estaría integrado de forma permanente en el vehículo y utilizaría el interfaz remoto de conexión. Además se modificaría la aplicación del móvil para actuar como pantalla adicional del segundo ordenador de abordo.
- Integrar una cámara trasera y un detector de proximidad en el vehículo y añadir a la aplicación la funcionalidad para utilizarlos. Desde el móvil se vería la imagen de la cámara al dar marcha atrás y también indicaría la distancia que queda con el obstáculo más cercano.

[1] El vehículo viene equipado de serie con un ordenador de abordo denominado OBC (*On Board Computer*). En él se pueden ver algunos parámetros del vehículo como el consumo, temperatura exterior, hora, alcance en kilómetros con el combustible que hay en el depósito, cronómetro, aviso al sobrepasar la velocidad indicada, etc.

Capítulo 9

Planificación y presupuesto

9.1. Planificación

En esta sección se detalla la planificación del proyecto. Por un lado se muestra una tabla con la lista de tareas y el tiempo dedicado a cada una de ellas, y por otro lado se muestra un diagrama de Gantt que detalla el listado anterior.

Hay que tener en cuenta que el esfuerzo dedicado no ha sido a tiempo completo al estar el personal involucrado en más proyectos. Se estima que la dedicación ha sido de una media de 3 horas al día.

9.1.1. Listado de tareas

En la tabla 9.1 se listan las tareas del proyecto, agrupadas o no según el caso, indicándose la fecha de inicio, la fecha de fin, y la duración en días.

Tarea	**Fecha de inicio**	**Fecha fin**	**Duración (días)**
Análisis del sistema	9/1/12	16/01/12	6
Estudio de la comunicación con el vehículo	17/1/12	23/3/12	49
Interfaz remoto	26/3/12	14/5/12	36
Diseño	26/3/12	23/4/12	21
Implementación	24/4/12	14/5/12	15
Aplicación en Android	15/5/12	20/6/12	27
Diseño	15/5/12	25/5/12	9
Implementación	28/5/12	20/6/12	18
Integración y pruebas	21/6/12	27/6/12	5
Documentación	9/1/12	28/6/12	124

Cuadro 9.1: Planificación del proyecto.

9.1.2. Diagrama de Gantt

En la figura 9.1 se muestra el diagrama de Gantt de las tareas listadas en el apartado anterior.

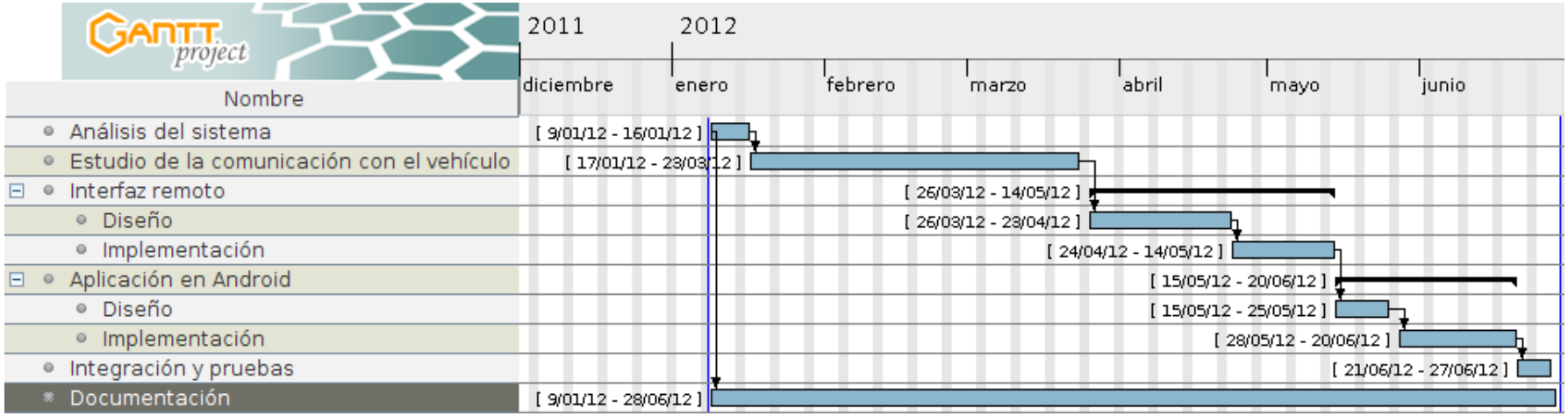


Figura 9.1: Diagrama Gantt de la planificación del proyecto.

9.2. Presupuesto

En esta sección se detalla el presupuesto para el presente proyecto según la guía de presupuestos para proyectos fin de carrera de la Universidad Carlos III de Madrid [15].

9.2.1. Autor

El autor del presente presupuesto es el ingeniero proyectista Francisco Olcina Grande.

9.2.2. Departamento

El presupuesto pertenece al Departamento de Informática de la Universidad Carlos III de Madrid.

9.2.3. Descripción del proyecto

El presente proyecto presenta un sistema que permite controlar de forma remota diferentes elementos de un vehículo mediante un móvil con sistema operativo Android.

9.2.4. Presupuesto total del proyecto

El presupuesto total para el desarrollo del proyecto asciende a DIEZ MIL SEISCIENTOS TREINTA Y SEIS EUROS CON VEINTICUATRO CÉNTIMOS DE EURO.

9.2.5. Desglose presupuestario (costes directos)

Personal

Para calcular los gastos asociados al personal se han tenido en cuenta las siguientes consideraciones:

- Se ha dedicado una media de 3 horas diarias al proyecto.
- No se contabilizan los siguientes días festivos:
  - 19 de Marzo del 2012.
  - 5 y 6 de Abril del 2012.

- 1 y 2 de Mayo del 2012.

La tabla 9.3 refleja los días reales trabajados por el personal a cargo del proyecto según los días no contabilizados del listado anterior.

Tarea	**Días reales**
Análisis del sistema	6
Estudio de la comunicación con el vehículo	48
Interfaz remoto	32
Diseño	19
Implementación	13
Aplicación en Android	27
Diseño	9
Implementación	18
Integración y pruebas	5
Documentación	119

Cuadro 9.2: Esfuerzos del personal del proyecto.

Con los siguientes datos acerca de la dedicación se puede obtener el coste asociado al proyecto de personal:

- Total días = 119
- Horas al día = 3
- Dedicación hombre/mes) 131,25 horas
- Coste hombre/mes = 2.694,38 €

A través de la siguiente fórmula se obtiene el coste:

$$Coste\ de\ personal = \frac{Total\ días\ \ x\ \ Horas/día}{Dedicación\ \ hombre/mes} * Coste\ hombre/mes$$

Como resultado se obtiene que el gasto de personal del proyecto asciende a SIETE MIL TRESCIENTOS VEINTIOCHO EUROS CON SETENTA Y UN CÉNTIMOS DE EURO. La tabla 9.3 muestra el personal asociado al proyecto.

Apellidos y nombre	Categoría	Días trabajados	Horas al día	Coste (en €)
Olcina Grande, Francisco	Ingeniero	119	3	7.328,71

Cuadro 9.3: Personal a cargo del proyecto.

Equipos

Los siguientes equipos informáticos han sido utilizados para la realización del proyecto:

- Pentium IV, adquirido con anterioridad y amortizado completamente.
- Portátil Acer Aspire One Happy, adquirido para el proyecto, con un coste de 280 €(iva incluido).
- Teléfono Sony Ericsson Xperia Neo V, adquirido para el proyecto, con un coste de 190 €(iva incluido).

La tabla 9.4 muestra la amortización de los equipos adquiridos para el desarrollo del proyecto.

Descripción	Coste sin IVA (euros)	% Uso dedicado al proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable
Portátil Acer Aspire One Happy	229,6	100	6	60	22,96
Teléfono Sony Ericsson Xperia Neo V	155,8	100	1	60	15,58
				Total	38,54

Cuadro 9.4: Amortización de equipos.

La formula utilizada para el cálculo de la amortización es la siguiente:

$$Cálculo\ de\ la\ amortización = \frac{A}{B} * C * D$$

Donde:

- A = número de meses desde la fecha de facturación en que el equipo es utilizado.
- B = periodo de depreciación (60 meses).

- C = coste del equipo (sin IVA).
- D = % de uso que se dedica al proyecto.

Como resultado se obtiene que el gasto de equipos del proyecto asciende a TREINTA Y OCHO EUROS CON CINCUENTA Y CUATRO CÉNTIMOS DE EURO.

Otros costes directos

La tabla 9.5 refleja una relación de otros costes directos imputables al proyecto. La suma de dichos costes asciende a CIENTO CUARENTA Y CUATRO EUROS CON VEINTITRÉS CÉNTIMOS DE EURO.

Concepto	**unidades**	**Coste unidad sin IVA (euros)**	**Coste total sin IVA (euros)**
Módem Bluetooth BlueSMiRF	1	40,17	40,17
Conector OBD I de 20 pines a OBD II	2	15,01	30,02
Centralita ZKE de desguace	1	16,4	16,4
Microcontrolador AVR ATMega644A	4	1,64	6,56
Placa protoboard	3	8,2	24,6
Placa cobre para PCB	1	9,84	9,84
Caja de plástico para PCB (9x15cm)	1	4,1	4,1
Diversos componentes electrónicos (resistencias, condensadores, zócalos, transistores,...)	1	12,54	12,54
		Total	144,23

Cuadro 9.5: Costes directos adicionales del proyecto.

9.2.6. Resumen de costes

La suma de todos los costes asociados al proyecto queda configurada según la tabla 9.6.

Concepto	**Importe**
Personal	7.328,71
Amortización de equipos	38,54
Otros costes directos	144,23
Costes indirectos (20 %)	1.502,29
Total sin IVA	9.013,77
Total con IVA (18 %)	10.636,24

Cuadro 9.6: Resumen de costes del proyecto.

El presupuesto total para el desarrollo del proyecto asciende a DIEZ MIL SEISCIENTOS TREINTA Y SEIS EUROS CON VEINTICUATRO CÉNTIMOS DE EURO.

Fuenlabrada a 29 de Junio del 2012

El ingeniero proyectista

Fdo. Francisco Olcina Grande

Anexo A - Código del microcontrolador AVR

```
#include <avr/io.h>
#include <avr/interrupt.h>


/*
   USART0 - Conexión al vehículo
   -----------------------------

   9.600 baudios
   8 bits de datos
   Paridad par
   2 bit de stop
*/
void USART_0_Init( unsigned int baud )
{

        /* Configuración de los baudios por segundo (9.600 bps) */
        UBRR0H = (unsigned char)(baud>>8);
        UBRR0L = (unsigned char)baud;

        //U2Xn: USART con velocidad de transmisión normal
        UCSR0A |= (0<<U2X0);

        /* Habilitado el envío y la recepción */
        UCSR0B = (1<<RXEN0)|(1<<TXEN0);

        /* Configuración del formato de transmisión:
         * 8 bits de datos, 2 bits de stop */
        UCSR0C = (1<<USBS0)|(1<<UPM01) | (1<<UCSZ01) |(1<<UCSZ00);


}

void USART_0_Transmit( unsigned char data )
{

        /* Espera a que esté vacio el buffer de envío */
        while ( !( UCSR0A & (1<<UDRE0)) );

        /* Introduce los datos en el buffer y los envía */
        UDR0 = data;

}

unsigned char USART_0_Receive( void )
{
        /* Espera a que haya datos para recibir */
        while ( !(UCSR0A & (1<<RXC0)) );

        /* Retorna los datos recibidos en el buffer */
        return UDR0;
}
```

```
/*
   USART1 - Conexión al modem bluetooth
   ----------------------------

   9.600 baudios
   8 bits de datos
   Paridad par
   2 bit de stop
*/
void USART_1_Init( unsigned int baud )
{
        /* Configuración de los baudios por segundo (9.600 bps) */
        UBRR1H = (unsigned char)(baud>>8);
        UBRR1L = (unsigned char)baud;

        //U2Xn: USART con velocidad de transmisión normal
        UCSR1A |= (0<<U2X1);

        /* Habilitado el envío y la recepción */
        UCSR1B = (1<<RXEN1)|(1<<TXEN1);

        /* Configuración del formato de transmisión:
         * 8 bits de datos, 2 bits de stop */
        UCSR1C = (1<<USBS1)|(1<<UPM11) | (1<<UCSZ11) |(1<<UCSZ10);
}

void USART_1_Transmit( unsigned char data )
{
        /* Espera a que esté vacio el buffer de envío */
        while ( !( UCSR1A & (1<<UDRE1)) );

        /* Introduce los datos en el buffer y los envía */
        UDR1 = data;
}

unsigned char USART_1_Receive( void )
{
        /* Espera a que haya datos para recibir */
        while ( !(UCSR1A & (1<<RXC1)) );


        /* Retorna los datos recibidos en el buffer */
        return UDR1;
}

int main (void)
{
        //Baud rate de 9.600 baudios con normal speed
        //Según tablas UBRRnH y UBRRnL => 23

        USART_0_Init (23);
        USART_1_Init (23);

        unsigned char byte_aux;

            while(1){

                    //Lee un byte del modem Bluetooth
                    byte_aux = USART_1_Receive();

                    //Lo escribe en el interfaz ADS
                    USART_0_Transmit (byte_aux);

            }

            return 0;
}
```

Glosario de términos

Activity	Componente de una aplicación de Android.
ADS	*Aktive DiagnoseStecker* o interfaz de diagnosis.
ADT	*Android Developer Tools* o plugin de desarrollo en Android para Eclipse.
Android	Sistema operativo para dispositivos móviles.
API	*Application Programming Interface* o interfaz para la programación de aplicaciones.
AVR	Microcontrolador de la compañía ATMEL.
ATMEL	Compañía conocida por la fabricación de microcontroladores.
Bluetooth	Sistema de comunicaciones inalámbrico.
BMW	Fabricante de automóviles.
Bonding	Tipo de enlace de dispositivos Bluetooth.
CAN Bus	*Controller Area Network* Bus o protocolo de comunicaciones basado en una topología bus.
CARB	(*California Air Resources Board* o agencia medioambiental de California.
Carsoft	Sistema de diagnosis para vehículos BMW y Mercedes.
Centralita	Unidad electrónica de un vehículo.
COM	Puerto serie RS-232C de un ordenador.
CPLD	Circuito lógico programable complejo.
CRC	Comprobación de redundancia cíclica.
CTS	Señal de *Clear To Send* del estándar RS-232C.
Dalvik	Máquina virtual utilizada por las aplicaciones de Android.
D-bus	Bus de diagnosis de los vehículos BMW.

DIP	*Dual in-line package*. Encapsulado para componentes electrónicos.
Discovery	Modo de operación de un dispositivo Bluetooth.
DTR	*Data Terminal Ready* o terminal de datos preparado.
E34	Vehículo BMW serie 5 fabricado entre 1989 y 1995.
Echo	Proceso en el cual se repite la cadena de texto anterior.
ECM	*Engine Control Module* o Módulos de Control de Motor.
ECU	*Electronic Control Unit* o centralita.
Ediabas	*Elektronik DIAgnose BASissystem* o sistema básico de diagnosis electrónica. Software de diagnosis utilizado por BMW.
EDR	*Enhanced Data Rate* o mayor velocidad de transmisión de datos.
FHSS	*Frequency Hopping Spread Spectrum* o espectro amplio de saltos de frecuencia.
Framework	Marco de aplicación o conjunto de bibliotecas orientadas a la reutilización de componentes software para el desarrollo de aplicaciones.
Full duplex	Canal de comunicación donde el receptor y el emisor transmiten al mismo tiempo.
GCC	*GNU Compiler Collection* o Colección de Compiladores GNU.
GND	*Ground* o tierra.
Half duplex	Canal de comunicación donde el receptor y el emisor pueden transmitir pero no simultáneamente.
I-bus	Tipo de bus presente en vehículos BMW.
Intent	Elemento existente en las aplicaciones para Android que indica la intención de realizar una acción.
ISO 9141	Normativa de diagnosis de vehículos.
ISO 9141-2	Normativa de diagnosis de vehículos.
ISO 14230 (KWP2000)	Normativa de diagnosis de vehículos.
ISP	*In-System Programming* o programación dentro del sistema.
(Línea) K	Línea de bus de datos de algunos vehículos.
K-bus	Tipo de bus presente en vehículos BMW.
(Línea) L	Línea de bus de datos de algunos vehículos.
MAC	*Media Access Control* o dirección de acceso al medio.
Manifest	Fichero de una aplicación de Android que proporciona información esencial sobre la misma.

M-bus	Tipo de bus presente en vehículos BMW.
Microcontrolador	Circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria.
MIL	*Malfunction Indicator Lamp* o luz indicadora de fallo.
MIPS	Millones de instrucciones por segundo.
OBD	*On Board Diagnostics* o sistema de diagnóstico a bordo de vehículos.
Pairing	Tipo de enlace de dispositivos Bluetooth.
P-bus	Tipo de bus presente en vehículos BMW.
PCB	*Printed Circuit Board* o circuito impreso.
Piconet	Tipo de red de dispositivos Bluetooth.
PIN	Código de seguridad de un dispositivo.
Pipe	Transferencia secuencial de información entre dos procesos.
Protoboard	Placa de pruebas para desarrollos electrónicos.
RI	*Ring Indicator* o indicador de llamada entrante.
RS-232C	Estándar de comunicaciones serie.
RTS	*Request to Send* o petición de envío.
RXD	Recepción de datos.
SAE	Sociedad de Ingenieros Automotrices.
SAE J1850	Normativa de diagnosis de vehículos.
SAE J1979	Normativa de diagnosis de vehículos.
Sandbox	Entorno aislado de ejecución de software utilizado para no corromper el sistema principal.
Scatternet	Tipo de red de dispositivos Bluetooth.
SDK	*Software Development Kit* o kit de desarrollo de software.
SGBD	del alemán SeBeschreibungsDatei o fichero descriptor.
Simplex	Canal de comunicación donde la información sólo fluye en un sentido entre el emisor y el receptor.
SMD	*Surface Mount Technology* o tecnología de montaje superficial.
Sniffer	Método de captura de información.
SOIC	*Small-outline integrated circuit*. Encapsulado para componentes electrónicos.
ToolSet 32	Software de diagnosis de BMW.

TTL	Estándar de transmisiones serie.
TXD	Envío de datos.
UART / USART	*Universal Asynchronous (Synchronous) Receiver/Transmitter* o transmisor/receptor universal asíncrono (o síncrono).
Vano del motor	Espacio del vehículo donde reside el motor.
VCC	Voltaje en corriente continua.
VIN	Número de bastidor de un vehículo. En los vehículos BMW se suele utilizar para referenciar sólo a los últimos 7 números del bastidor.
ZKE / ZVM	*Zentrale Karosserie Elektronik* o *Central Body Electronics* o *General Module*. Centralita de un vehículo BMW encargada de los elementos de la carrocería.

Bibliografía

[1] BMW AG. *Dokumentation zum Aktiven Diagnose Stecker*. 1997.

[2] BMW AG. *EDIABAS, User manual installation guide*. version 6c.

[3] BMW AG. *New General Modules (GM)*. Service Information Bulletin supersedes S.I. 61 01 92 (3460). Woodcliff Lake, NJ, 1992.

[4] ATMEL. *ATMEL Download AVR Studio 4.19*. URL: http://www.atmel.com/dyn/resources/prod_documents/AvrStudio4Setup.exe.

[5] ATMEL. *Datasheet ATMEL, 8-bit AVR Microcontroller (ATmega644A)*. Revisión 8272A. 2010.

[6] "Benemorius". *(Almost) free DIY ADS interface*. 2011. URL: http://forums.bimmerforums.com/forum/showthread.php?t=1650885.

[7] BMW. *BMW 525i, 525it, 535i, M5 (E34) Electrical Troubleshooting Manual*. 1993.

[8] *BMW Bus System Troubleshooting*. URL: http://www.e38.org/bussystem.pdf.

[9] California Air Resources Board. *On-Board Diagnostic II (OBD II) Systems - Fact Sheet / FAQs*. 2009. URL: http://www.arb.ca.gov/msprog/obdprog/obdfaq.htm.

[10] B&B Electronics. *OBD-II Background*. URL: http://www.obdii.com/background.html.

[11] Wikipedia (english). *RS-232*. URL: http://en.wikipedia.org/wiki/RS-232.

[12] Mark Gibson. *BMW Data Link Interface Project, DS2 Protocol*. URL: http://markgardnergibson.com/BMW/protocol.html.

[13] "HansV". *(Allmost) free OBD interface*. 2012. URL: http://forums.bimmerforums.com/forum/showpost.php?p=23874562&postcount=77.

[14] André N. Klingsheim. *Master's Thesis, J2ME Bluetooth Programming*. Capítulo 2 (Bluetooth). 2004.

[15] Universidad Carlos III de Madrid. *Plantilla para presupuestos de Proyectos Fin de Carrera*. URL: http://www.uc3m.es/portal/page/portal/administracion_campus_leganes_est_cg/proyecto_fin_carrera/Formulario_PresupuestoPFC-TFG%20(3)_1.xlsx.

[16] Autoxuga Móvil. *Antecedentes de los Escáneres*. URL: `http://www.autoxuga.com/html/escaneresantecedentes.htm`.

[17] Aldo Nicolás Bianchi Petrone. *REDES DE COMUNICACIONES, Transmisión de Datos, Redes de Computadores y Redes Digitales de Comunicaciones*. Capítulo 3. 2002.

[18] Bentley Publishers. *BMW 5 Series Service Manual 525i,530i,535i,including Touring, 1989,1990,1991,1992,1993,1994,1995*. 1998.

[19] Erik Svedbäck Pär Enström. *Master's Thesis, Road Vehicle Diagnostics using Bluetooth*. Capítulo 2 (Communication Mediums). 2003.

[20] Sparkfun. *Bluetooth Modem - BlueSMiRF Silver*. URL: `http://www.sparkfun.com/products/10269`.

[21] Andrew S. Tanenbaum. *Redes de computadoras*. Cuarta edición. 2003.

[22] Wikipedia. *Android*. URL: `http://es.wikipedia.org/wiki/Android`.

[23] Wikipedia. *AVR*. URL: `http://es.wikipedia.org/wiki/AVR`.

[24] Wikipedia. *Bluetooth*. URL: `http://es.wikipedia.org/wiki/Bluetooth`.

[25] Wikipedia. *MAX232*. URL: `http://es.wikipedia.org/wiki/MAX232`.